Treghetsnavigasjon implementert i Android

Sami Hamra

Master oppgave ved Fysisk institutt

UNIVERSITETET I OSLO

27/06/2011

Forord

Jeg vil takke min Internveileder, Oddvar Hallingstad for å gi meg tilgang til sin mobiltelefon, og ikke minst hans hjelp og råd om framdrift i mitt arbeid og min rapport.

Jeg vil også takke mine venner, kolleger og familie for deres støtte gjennom perioden.

Kjeller, 27.juni 2011

Sami Hamra

Universitet i Oslo

Sammendrag

Mobiltelefoner med 3-aksede gyroskoper og akselerometre er nå tilgjengelig på markedet. Dette gjør det mulig å implementere et treghetsnavigasjonssystem (TNS) i telefonen som blant annet kan brukes til å måle volumet av 3-dimensjonale objekter. I den forbindelsen har det blitt utvikletmobiltelefon-applikasjon som logger data fra sensorer (akselerometer og Gyroskop) på en Nexus S. Disse måledataene har blitt implementert i Matlab og brukt videre for å se på brukbarheten til disse dataene.

Det har blitt utviklet matematiske modeller for det fysiske- og mekaniserte system for mobiltelefonen.Det mekaniserte systemet, som også kalles Navigasjonssystemet, løser den matematiske modellen av det fysiske systemet ved bruk av målt spesifikk kraft og målt vinkelhastighet. Dette systemet består av navigasjonslikninger for treghetsnavigasjon på flat ikke- roterende jord. Navigasjonslikningene bruker de rå måledataene fra sensorene til å beregne posisjon, hastighet og orientering for mobiltelefonen ved bruk av integrasjonsrutiner.

Grunnet mangel påmobiltelefonens sanne spesifikk kraft, og vinkelhastighet fra akselerometeret og gyroskopet kunne ikke sann- posisjon, hastighet, og orientering beregnes.

Tre forskjellige sett av måledata ble tatt med:

1. Mobiltelefonen i ro uten rotasjon.
2. Mobiltelefonen i ro med 90 graders rotasjon rundt z-aksen.
3. Kvadratisk bevegelse

Analyser av beregnet posisjonen, hastighet og orientering i alle retninger ved bruk av målte spesifikk kraft, og målte vinkelhastigheter gjort. Resultatene viser at sensorene blir påvirket av støy, og andre feilkilder. For en komplett analyse av brukbarheten av sensorene må et kalmanfilter implementeres.

Innholdsfortegnelse

Figurer

Tabeller

1 Introduksjon

Som følge av miniatyrisering av elektroniske komponenter, har datamaskinene blitt en større del av vårt daglige liv.Datamaskiner som før tok mye plass i store rom kan i dag oppbevares i lommer. Mobiltelefon er uten tvil den mest populære av alle håndholdte enheter, og blir tatt med nesten overalt eieren reiser. Selv om mobiltelefonen hovedsakelig brukes til å ringe og sende meldinger, har plattformen et potensiale til å brukes til mye mer avanserte formål.

Formålet med denne oppgaven er å utvikle en applikasjon for en mobiltelefon som skal samle inn måledata fra sensorer (akselerometer og Gyroskop) på en mobiltelefon(Android Nexus S). Disse måledataene skal implementeres og brukes videre i Matlab hvor man skal se på brukbarheten av disse. Målsettingen med oppgaven er å finne hvilken grad av nøyaktighet man kan oppnå med telefoner som har innebygdesensorer som kan bli satt sammen til et treghetsnavigasjonssystem(TNS). TNS brukes både i sivil og militær luftfart. Mobiltelefoner som har 3-aksede gyroer og 3-aksede akselerometre er nå tilgjengelige på markedet. Dette gjør det mulig og implementere et TNS i telefonene som blant annet kan brukes til å måle volumet av 3-dimensjonale objekter. Dette kan ha stor nytteverdi hos de fleste fraktselskapene. Hovedkomponentene i et TNS er sensorer og dataprogrammer som løser navigasjons- og kalmanfilter likninger.

1.1 Treghetsnavigasjon

Treghetsnavigering er en måte å navigere på ved bruk av en tidligere kjent posisjon og bevegelse som har forekommet i ettertid. Denne typen navigasjon tar ikke i bruk eksterne referansepunkter som for eksempel*Global positioning system*(GPS) osv., men kan allikevel med en viss nøyaktighet gi den nåværende posisjonen.

Hovedkomponenten i et treghetsnavigeringssystem (TNS) erTreghets Måle Enhet (TME) også kalles*Initial Measurment Unit* (IMU). Dette er en elektronisk enhet som bruker akselerometre, og gyroskoper til å måle- og rapportere hastighet, orientering og gravitasjonskrefter.

Et treghetsnavigasjonssystem er bygget opp for å måle enhetens vinkelhastighet og den spesifikke kraften som deretter integreres opp for å gi orientering, fart, og posisjon. En utfordring er at målinger ofte blir påvirket av feilkilder og støy. Dette betyr at det er vanskelig å få nøyaktige målinger av vinkelhastighet og spesifikk kraft, som igjen kan medføre betydelige feil i integrasjonsalgoritmen. For å unngå for store unøyaktigheter bruker man matematiske modeller til korrigering av målingene,som deretter kan kjøres gjennom et kalmanfilter.

1.2 Kalmanfilter

Et Kalmanfilter (KF) er en matematisk metode oppkalt etter Rudolf E. Kalman. KF kan bli forklart ved følgende analogi.En mann som sover i et mørkt rom, som han kjenner, våkner opp midt på natta og vil gå ut av rommet.Lysene på rommet er av, og derfor må han finne utgangsdøren i totalt mørke.Han har en a *priori* informasjon om hvor døren er plassert, og utfører etestimat basert på denne informasjonen, og begynner å gå i denne retningen. Etter kort tid går han rett på en vegg. Dette betraktes som en måling. Nå vet han litt mer hvor han befinner seg i rommet. Med andre ord han oppdaterer sitt estimat med denne målingen. På dette punktet er en prediksjon av en ny retning mulig. Basert på hans kjennskap av rommet er det nå mulig å foreta en ny prediksjon ved dette punktet. Prosessen gjentas med flere målinger, og vil dermed øke sjansene hans for å finne døren. Formålet med et kalmanfilter er

å bruke målinger som er observert over tid, som inneholder støy(tilfeldige variasjoner) og andre unøyaktigheter, og produsereverdier som er nærmere de sanne verdiene av målingene.

I denne oppgaven brukes kalmanfilteret for å redusere tilfeldig støy i akselerometer -og gyroskop dataene. Kalmanfilteret kan brukes til å kombinere målinger fra flere sensorer samtidig, noe som gir både et estimat av nåværende tilstand i et system, og en prediksjon av fremtidig tilstand av systemet.

1.3 Android

Android er en*open-source* programvareplattform for håndholdte enheter utviklet av Google. Plattformen inneholder et operativt system, *middleware*[1], og andre Java-baserte nøkkel programmer.

Det finnes mange håndholdte enheter i dag som er laget av ulike produsenter, men alle er basert på Android. Dette betyr at *interfacet* både for utviklere og brukere er tilnærmet den samme uavhengig av produsent. Applikasjoner som er utviklet for en bestemt enhet skal med andre ord enkelt kunne brukes på en annen enhet. Android sittoperativsystem er verdens mest solgte smarttelefon plattform.To nøkkel- komponenter i en hver programvare utvikling er IDE, og SDK.

1.3.1 EclipseIDE

Integrated Development Environment (IDE) også kjent som *Integrated Design Environment*eller,*Integrated Debugging Environment* er en Software applikasjon som gir omfattende fasiliteter til datamaskin programmer for programvareutvikling. En IDE består normalt av [1]:

[1]Middleware er software som håndterer interaksjon mellom ulike prosesser/software komponenter. Et eksempel kan være interaksjon mellom en nettleser og nettverkskortet på enheten nettleseren kjøres. Mange av dagens operativ systemer har mye av det man tidligere så på som middleware ferdig integrert.

- Kode editor
- Kompiler
- Debugger

Det finnes flere varianter av IDE. Eksempler på disse er Eclipse og NetBeans.

Eclipse er skapt av en *opensourcecommunity*, og brukes i flere ulike områder. I denne oppgaven brukesEclipse for å logge data fra sensorer, og lagre disse i SD kortet. IDE er primært skrevet i JAVA, men i Eclipse finnes det proviantfor å installere en tilpasset Plug-in kalt *Android Development Tools*(ADT) som støtter oppretting, drift, og feilsøking av Android applikasjoner.

1.3.2 AndroidSDK

Siden Android plattformen er *open-source* betyr dette at hvem som helst kan utvikle Software løsninger til den. Et*Software Development Kit* (SDK) gis ut for hver versjon av plattformen.

Det finnes mange komponenter som er tilgengelige for SDK'etsom for eksempel *SDK Tools*, SDK plattformer, og USB driver for Windows. I denne oppgaven ble de nødvendige basiskomponentene lastet ned og installert som anbefalt på Androids hjemmeside. Under følger det en liste over de komponentene som ble lastet ned og brukt[2]:

- SDK Tools
  Det er nødvendig å ha SDK Tools for å utvikle et Android-program.

- SDK Plattform-Tools
  Dette inkluderer flere verktøy som kreves for applikasjonsutviklingen. Disse verktøyene er plattform-avhengige, og oppdateres vanligvis bare når en ny SDK plattform er tilgjengelig for å støtte nye funksjoner i plattformen.

- SDK plattform

Man må laste ned minst en plattform, slik at man kan kompilere applikasjonen, og sette opp en *AndroidVirtula Device*(AVD) for å kjøre applikasjonen på emulatoren.

1.3.3 Hardware

Når man skal lage en applikasjon, er det viktig at man alltid tester programmet på en ekte enhet før man kan legge det ut til brukere. Det er kun mulig å bruke Android-drevne enheter for å kunne kjøre, feilsøke og teste programmet. Verktøyene i SDK'etgjør det enkelt å installere- og kjøre programmet på enheten hver gang man kompilerer. Man kan enkelt installere programmer på enheten direkte fra Eclipse, eller fra kommandolinjen med ADB. Kravene til hardware[3] når det gjelder denne oppgaven er:

- 3- akse akselerometer
- 3-akse gyroskop
- 3-akse Magnetisk felt

For å kunne kjøre programmet på en enhet trenger man å aktivere et par ting på enheten først. Under innstilinger i enheten må man tillate- *Unknownsorce*, og *Debuging*. Nexus S er en ren google telefon, det betyr at det ikke finnes produsent spesifikk software på telefonen, noe som gjøre den egnet til utvikling.

1.4 Modellering

Formålet med denne oppgaven er å samle måledata fra sensorer som ligger i en mobiltelefon. Disse måledataene skal bli implementert i Matlab hvor man skal se på nøyaktigheten, og brukbarheten av disse måledataene. For å kunne jobbe med og analysere disse dataene, må matematiske modeller som beskriver systemet, feilsystemet, og Kalmanfilter likningene bli satt opp. Modelleringene vil bli vist i kapitel 2 i rapporten.

2 Teori

I denne oppgaven skal man utvikle et program som samler måledata fra sensorer (akselerometer, Gyroskop, og Kompass) fra en mobiltelefon (Nexus S). I dette kapittelet vil det bli gitt en innføring i teori som resten av oppgaven forutsetter at leseren har kjennskap til.

2.1 Koordinatsystemer og relasjoner mellom disse

For å sette opp TNS- og Kalmanfilter likningene må man definere konstantene i systemet, koordinatsystemene og relasjonene mellom disse.

Konstanter

Alle enhetene er SI-enheter:

$$g = 9.81\, m/s^2$$

Initial verdiene:

$$p_0^n = [0; 0; 0] \tag{2.1}$$

$$\omega_0^n = [0; 0; 0] \tag{2.2}$$

$$g^n = [0; 0; -g] \tag{2.3}$$

2.1.1 Rammer

I navigasjon, kan flere referanserammer brukes til å presentere data. Avhengig av hva navigasjonssystemet brukes til å få målinger, er forskjellige referanse systemer vanligvis nødvendige. Følgende rammer (koordinatsystemer) ble brukt:

- $\{n\} = \{0_n;\ \vec{x}_n;\ \vec{y}_n;\ \vec{z}_n\}$: Treghetsramme er ikke roterende i forhold til stjernene og har origo i jordas sentrum, $\vec{z}_n$peker nordover og 0_n ligger i $\vec{z}_n$-$\vec{x}_n$ planet ved t= 0.

- $\{b\} = \{0_b;\ \vec{x}_b;\ \vec{y}_b;\ \vec{z}_b\}$: Legemeramme antas her å ligge fast i mobilen med origo i treghetsplattformen

2.1.2 Koordinattransformasjonsmatriser

For å transformere fra en ramme til en annen, kan rotasjonsmatriser brukes[4]. For en rotasjonsmatrise, senket bokstavene indikerer rammen den blir transformert fra, mens hevet skrift betegner rammen den blir transformert til. På den måten kan man skrive (R_b^n), hvor *b* rammen er den ramme man transformerer fra og *n* rammen er den rammen man transformerer til. De tre elementære RKM er definert ved:

$$R_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.4}$$

$$R_2(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.5}$$

$$R_3(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

RKM R_b^n kan uttrykkes vha 3-2-1 eulervinkleneθ_1, θ_2 og θ_3(subskriptet angir hvilken akse dreiningen er om):

$$R_b^n(\underline{\theta}) = R_3(\theta_3)R_2(\theta_2)R_1(\theta_1) \tag{2.7}$$

$$\tilde{R}_b^n = R_b^n(\underline{\theta}) \tag{2.8}$$

Målt vinkelhastighet og spesifikk kraft

Målt vinkelhastighet($\underline{\tilde{\omega}}_b^n$) fås fra gyroskopsmålingene og målt spesifikk kraft ($\underline{\tilde{f}}^n$) fås fra akselerometersmålingene.

2.2 TreghetsMåle Enhet (TME)

Et rent TNS består kun av akselerometre og gyroskoper, og er basert på prinsippet om at estimatene av posisjonen og hastigheten oppnås ved å integrere akselerasjonen. Generelt,TME'er [5]er svært kostbare, på grunn av behovet for svært nøyaktige målinger. Grunnen til at nøyaktigheten av målingene er nødvendig, er at akselerasjonen er integrert to ganger for å få posisjonen. Eventuelle feil i akselerasjonsmålingene vil også bli integrert, og føre til en Bias feil i den estimerte hastigheten, og en kontinuerlig drift på posisjonsestimat, med mindre den blir korrigert. Korrigering på et rent TNS er umulig, derfor bruker man et kalmanfilter for å gjøre denne jobben.

I denne oppgaven benyttes MEMS gyroskoper og MEMS akselerometre

Microelectromechanical systemer (MEMS) [6], er teknologien av svært små mekaniske enheter drevet av elektrisitet. Den flettes på nano-skala i*Nanoelectromechanical systems*(NEMS), og nano teknologi.

Android enheten har mer enn 12 ulike sensor typer, forskjellige applikasjoner krever forskjellige sensorer. I denne oppgaven skal det lages en applikasjon for å måle volumet av3-dimensjonale objekter. Det vil si at man trenger posisjonen, hastigheten og orientering til enhver tid.

Vi bruker notasjonene:

$$x(t) = posisjonvedtident$$

$$v(t) = hastighetenvedtident$$

$$a(t) = akselerasjonen\ ved\ tiden\ t$$

Knytter hastigheten til akselerasjonen

Akselerasjonen er definert av den deriverte, matematisk sett kan det skrives:

$$a(t) = \frac{\delta v}{\delta t} \tag{2.9}$$

Ved å bruke det grunnleggende Kalkulus teoremet kan man skrive dette forholdet på formen:

$$\int_{T_1}^{T_2} a(t)\ \delta t = v(T_2) - v(T_1) \tag{2.10}$$

Hvis man kaller start tiden t = 0, og slutt tiden t = T, er det mulig å skrive dette integralet som:

$$\int_{0}^{T} a(t)\ \delta t = v(T) - v(0) \tag{2.11}$$

Knytter posisjonen til hastigheten

Hastigheten er definert av den deriverte av posisjonen:

$$v(t) = \frac{\delta x}{\delta t} \tag{2.12}$$

Ved å bruke grunnleggende Kalkulus teoremet kan man skrive dette forholdet på formen:

$$\int_{T_1}^{T_2} v(t)\ \delta t = x(T_2) - x(T_1) \tag{2.13}$$

der forskjellen mellom slutt- og start posisjoner $x(T_2) - x(T_1)$, kalles forskyvingen.

Akkurat som tidligere kan man velge å starte klokken, slik at prosessen foregår mellom t = 0, og t = T, slik at integralet blir:

$$\int_{0}^{T} v(t)\ \delta t = x(T) - x(0) \tag{2.14}$$

Posisjoneringsteknologien kan grovt deles i to hoveddeler, relativ posisjonering og absolutt posisjonering. Absolutt posisjonering betyr at nåværende posisjon ikke er avhengig av tidligere posisjoner, mens relativ posisjonering betyr at nåværende posisjon er avhengig av tidligere posisjoner.

Et akselerometer måler spesifikk kraft, som er akselerasjon sett fra treghetsrammen. Ved å integrere dette får man hastigheten, og ved å integrere hastigheten får man posisjonen. I tillegg gir gyroskopet vinkelhastigheten. Ved å integrere denne får man vinkelen. Et digitalt kompass gir retningen mobilen beveger seg mot.De inngående sensorene i denne oppgaven er derfor akselerometer, gyroskop, og digital kompass (Magnatic Field).

2.2.1 MEMS Akselerometer

Et akselerometer [7] er en sensor som måler akselerasjonen av en enhet sett fra treghetsrammen.Akselerometeret har til hensikt å beregne av avstandsmålinger for mobiltelefon, eller plattform i kort varighet. Avstanden mobiltelefonen har beveget seg kan man finne ved dobbel integrasjon av måledataene fra akselerometersensoren med hensyn til tid. Biasfeilen i akselerasjonssignalet er akkumulert, og nøyaktigheten av avstandsmålingen kan svekkes med tiden på grunn av integrering. Dette problemet kan løses ved periodiskre-kalibrering med hjelp av eksterne målinger av posisjon, og hastighet. Disse eksterne signaler kan beregnes ved bruk av et TNS. I et kalmanfilter kan man bruke disse verdiene for å gi et optimalt estimat av systemetsfeilen. Med den rette kompensasjon av gravitasjonskrefter kan akselerometeret være en bra løsning som et avstands-måleinstrument for en mobiltelefon.Den spesifikke kraften målt fra akselerometeret kan defineres som:

$$\vec{f} = \vec{a}_b^n + \vec{g}^n \tag{2.15}$$

der $\vec{g}^n$ betegner posisjonen relative graviteten.

TNS'et måler krefter i sin nåværende posisjon, betegnetsom (f^b) har følgende egenskaper i dette tilfelle:

$$\underline{f}^b = R_n^b \underline{a}_b^n + R_n^b \underline{g}^n \tag{2.16}$$

Der$\underline{f}^b$ er sann spesifikk kraft, vet vi at:

$$\begin{aligned} \delta\underline{f} &= \underline{f}^b - \underline{\tilde{f}}^b \\ \delta\underline{f} &= \underline{\beta}^a + \underline{v}^a \end{aligned} \tag{2.17}$$

Hvor $\underline{\tilde{f}}^b$ er målt spesifikk kraft fra akselerometeret, kan vi skrive likningen for målt spesifikk kraft $\underline{\tilde{f}}^b$:

$$\underline{\tilde{f}}^b = \underline{f}^b + \underline{\beta}^a + \underline{v}^a \tag{2.18}$$

Akselerometeret har SI-enhetene meter per sekund i andre ($\frac{m}{s^2}$).

2.2.2 MEMS Gyroskop

Mye av utviklingen som har blitt gjort angående ytelsen til gyroskopene kan tilskrives The Charles Stark Draper Laboratories Inc, i USA. I de siste 50 årene har de funnet opp, og

utviklet TNS, som brukes på jorda og i verdensrommet. Laboratoriet har også utviklet MEMS gyroskop de siste 10 årene, og de var også først ute med å demonstrere måling av en rotasjons-rate ved bruk av en mikro-maskinert silikon sensor[6].Liten størrelse, lav vekt

Karakteristikken av liten størrelse, lav vekt, høy funksjonsstabilitet, og lav pris har tillatt MEMS gyroskoper til å finne plass i forbrukerelektronikk, robotikk, og bilindustri markeder, samt bruk som treghetsnavigering sensorer i luftfart og avionikk applikasjoner. Treghets MEMS gyroskop sensorer bruker vibrerende mekaniske elementer til å sanse rotasjon.

For en tre-akset gyro er den generelle målelikningen definert som[6]:

$$\underline{\widetilde{\omega}}_b^{nb} = \underline{\omega}_b^{nb} + \delta\underline{\omega} \tag{2.19}$$

Hvor $\underline{\widetilde{\omega}}_b^{nb}$ er målt vinkelhastighet fra gyroskopet, vi vetat $\delta\underline{\omega}$ kan skrives:

$$\delta\underline{\omega} = \underline{\omega}_b^{nb} - \underline{\widetilde{\omega}}_b^{nb}$$

Derved kan vi skrive likningen for målt vinkelhastighet

$$\underline{\widetilde{\omega}}_b^{nb} = \underline{\omega}_b^{nb} + \underline{\beta}^g + \underline{v}^g \tag{2.20}$$

Akkurat som akselerometre, er gyroskop en bevegelses detektor enhet. Dataene fra gyroskopet har SI-enheten radianer per sekund ($\frac{r}{s}$).

Akselerometreproduserer signaler i forholdtillineærbevegelse, mensgyroskoperproduserersignaleri forholdtilrotasjonsbevegelse. I denne oppgaven skal gyroskopet gi vinkelhatigheten til mobiltelefonen- mens den står i ro med 0 rotasjon, mens den står i ro med rotasjon på 90 grader og når mobiltelefonen er i kvadratisk bevegelse uten rotasjon. Gyroskoper er begrenset av ulike feilkilder som påvirker langsiktige og kortsiktige resultater, som for eksempel Bias feil.

2.3 Kompass

Magnetfelt sensorer er ofte brukt i industriell biomedisinsk felt. De er viktige sensorer når det gjelder måling av jordmagnetiskfelt, og magnetiskmønster avbilding. I de fleste applikasjoner bør magnetiskfelt sensorene være små, og ha lavt strømforbruk samtidig stille høye krav når det gjelder sensitivitet og nøyaktighet. Dessverre er presentasjonen av de fleste nåværende magnetfelt sensorer ikke tilfredsstillende[8]. Det har blitt installert og testet vanlige kompass applikasjoner fra Android markedet for å teste nøyaktigheten på magnetfelt sensoren som ligger i Nexus S. De testene har vist en stor unøyaktighet i denne sensoren, derfor har ikke den sensoren blitt brukt i denne oppgaven. Selv om denne sensoren ikke har blitt brukt i oppgaven,blir applikasjon for logging av data fra denne sensoren utviklet, og lest ut sammen med dataene fra akselerometer og gyroskop sensorene.

2.4 Sensor tilgjengelighet i AndroidSDK

Sensor Manageren i Android gir tilgang til enhetens sensorer. Mer detaljer om Sensor Manageren og hvordan man oppnår kommunikasjon mellom applikasjonen og sensorene forklares i kapitel 3.

2.5 TNS

Posisjonering kan bety mye, men i dette tilfelle gjelder det å beregne posisjonen etter en visstid, når man allerede vet startposisjonen. *Deadreckoning* (DR) [9]er prosessen for å estimere nåværende posisjon basert på en tidligere bestemt posisjon, eller ordne på den posisjonen basert på en tidligere bestemt hastighet med hensyn til tidog retning. Moderne treghetsnavigasjonssystemer (TNS) som er avhengige avDR er svært mye brukt.

2.5.1 Treghetsnavigasjon

Matematisk modell av mobilen uttrykkes ved hensyn på spesifikk kraft og vinkelhastigheten i treghetsramme {n}[4]:

$$\dot{\underline{p}}_b^n = \underline{v}_b^n \tag{2.21}$$

$$\dot{\underline{v}}_b^n = R_b^n \underline{f}^b - \underline{g}^n \tag{2.22}$$

$$\dot{\underline{\theta}} = \mathrm{D}(\theta)\underline{\omega}_b^{nb} \tag{2.23}$$

Hvor $\underline{\omega}_b^{nb}$ er sann vinkelhastighet, og f^b er sann spesifikk kraft. I dette tilfelle har vi ikke sann vinkelhastighet og spesifikk kraft, derfor må man sette opp navigasjonslikninger, og bruke de videre når man skal sette opp kalmanfilterlikningene.

2.5.2 Tre-akset plattform, flat ikke-roterende jord

Denne oppgaven er begrenset til å navigere på en flat ikke-roterende jord i tre dimensjoner [10].

Modell av fysisk system

Denne modellen beskriver sann systemmodell av det fysiske systemet.
Systemet er gitt ved følgende likninger:

$$\dot{\underline{x}}^n = \underline{v}^n \tag{2.24}$$

$$\dot{\underline{v}}^n = R_b^n \underline{f}^b - \underline{g}^n \tag{2.25}$$

$$\dot{\underline{\theta}} = \mathrm{D}(\theta)\underline{\omega}_b^n \tag{2.26}$$

På grunn av mangel på informasjon når det gjelder sann vinkelhastighet($\underline{\omega}_b^{nb}$), og sann spesifikk kraft($\underline{f}^b$) kan vi ikke bergene på det fysiske systemet.

Navigasjonslikninger

Navigasjonslikningene løser den matematiske modellen av fysiske systemet ved bruk av målt spesifikk kraft, og målt vinkelhastighet. Modellen kan beskrives med følgende likninger:

$$\dot{\underline{p}}_b^n = \underline{\tilde{v}}_b^n \tag{2.27}$$

$$\dot{\underline{v}}_b^n = \tilde{R}_b^n \underline{\tilde{f}}^b - \underline{g}^n \tag{2.28}$$

$$\dot{\underline{\theta}} = \mathrm{D}(\theta)\underline{\tilde{w}}_b^{nb} \tag{2.29}$$

Hvor

$$\mathrm{D}(\theta) = \begin{bmatrix} 1 & (\sin\theta_1 \tan\theta_2) & (\cos\theta_1 \tan\theta_2) \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \frac{\sin\theta_1}{\cos\theta_2} & \frac{\cos\theta_1}{\cos\theta_2} \end{bmatrix} \tag{2.30}$$

Dette systemet innebærer målte verdier spesifikk kraft ($\underline{\tilde{f}}^b$) og vinkelhastighet ($\underline{\tilde{w}}_b^{nb}$).

Løsningen av navigasjonslikningene gir posisjon, hastighet, og orientering. Tilstandsvektoren til navigasjonslikningene inneholder følgende komponenter:

$$\underline{\tilde{x}}^n = [\underline{p}^n \underline{v}^n \ \theta]^T \tag{2.31}$$

pådraget for dette systemet består av målt vinkelhastighet ($\underline{\tilde{w}}_b^{nb}$), og målt spesifikk kraft($\underline{\tilde{f}}^b$):

$$u(t) = \begin{bmatrix} \underline{\tilde{f}}^b \\ \underline{\tilde{w}}_b^{nb} \end{bmatrix} \tag{2.32}$$

I denne oppgaven vil vi først og fremst se på hvor nøyaktig akselerometer, og gyroskop måle verdiene er, derfor blir ikke støy og andre feilkomponenter tatt med her, men det blir tatt med senere under feilmodellen.

Modell av Feillikningene

Feilmodellen som beskriver systemet kan bli sett på som differansen mellom det fysiske systemet (sanne verdier) og den mekaniserte system (målte verdier), men siden vi ikke har sanne verdier for spesifikk krat og vinkelhastigheten kan man beskrive systemet basert på navigasjonslikningene som følgende:

$\delta\dot{\underline{x}} = F(\underline{\tilde{x}}) + G\underline{v}$, hvor F er systemmatrise, og G er prosesstøymatrisen

$$\delta\dot{p}^n = \delta v^n \tag{2.33}$$

$$\delta\dot{v}^n = -\mathrm{S}(\tilde{R}_b^n \underline{\tilde{f}}^b)\epsilon^n - \tilde{R}_b^n \beta^a - \tilde{R}_b^n v^a \tag{2.34}$$

$$\dot{\varepsilon}^n = -\tilde{R}_b^n \beta^g - \tilde{R}_b^n v^g \tag{2.35}$$

$$\dot{\beta}^a = 0 \tag{2.36}$$

$$\dot{\beta}^g = 0 \tag{2.37}$$

Feilmodellen ble beskrevet som differansen, og blir derfor presentert som deltaverdier. Akselerometeret og gyroskopet inneholder prosesstøy, og det blir presentert her som bias feil. Bias feilen er konstant, og derfor kan den deriverte bli satt til 0. β^a beskriver bias feilen i akselerometeret og β^gbeskriver bias feilen i gyroskopet.

Tilstandsvektoren for feilmodellen kan derved uttrykkes:

$$\delta x = [\delta p^n \delta v^n \epsilon^n \beta^a \beta^g]^T \tag{2.38}$$

Feilsystemet har ikke pådrag, det vil si at $u = 0$, mens prosesstøyen som påvirker feilsystemet inneholder bidrag fra både akselerometeret og gyroskopet og kan uttrykkes:

$$\underline{v} = [v^a v^g]^T \tag{2.39}$$

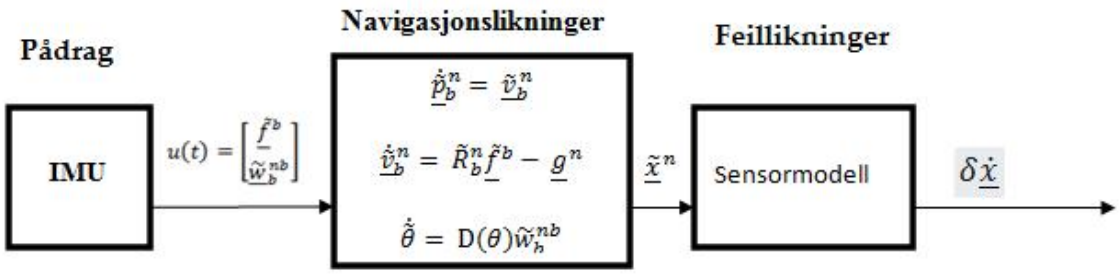
Vi vet at:

$\delta\dot{\underline{x}} = F(\underline{\tilde{x}}) + G\underline{v}$, derved kan man sette opp systemet matematisk som følgende:

$$\begin{bmatrix} \delta\dot{p}^n \\ \delta\dot{v}^n \\ \dot{\varepsilon}^n \\ \dot{\beta}^a \\ \dot{\beta}^g \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\mathrm{S}(\tilde{R}_b^n \underline{\tilde{f}}^b) & -\tilde{R}_b^n & 0 \\ 0 & 0 & 0 & 0 & -\tilde{R}_b^n \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta p^n \\ \delta v^n \\ \epsilon^n \\ \beta^a \\ \beta^g \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\tilde{R}_b^n & 0 \\ 0 & -\tilde{R}_b^n \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v^a \\ v^g \end{bmatrix} \quad (2.40)$$

$$\delta\underline{z_k} = H(\underline{x_k}) + \underline{w_k} \quad (2.41)$$

2.5.3 Blokkskjema for TNS'et



Figur 2.1: oppsummering av TNS i et blokkskjema

2.5.4 Kalmanfilterlikningene

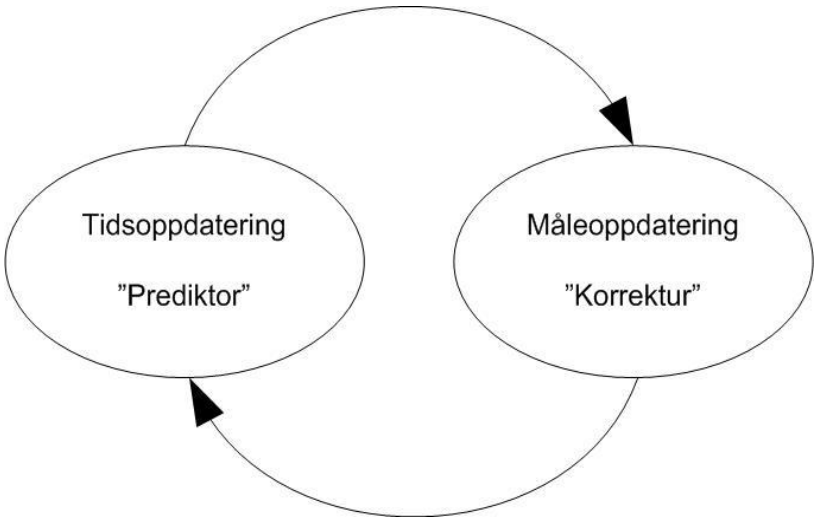
Et Kalmanfilter[4] blir brukt som en tilstandsestimator for å estimere tilstandene i et stokastisk lineært dynamisk system. Dette gjøres på grunnlag av målinger, pådrag og kjennskap til de stokastiske egenskapene til prosesstøy og målestøy. Ved bruk av et kalmanfilter kan man oppnå mange fordeler. Et godt eksempel på et område kalmanfilteret har gjort stor nytte, er i integrerte navigasjonssystemer. Her anvendes filteret for å oppnå et optimalt estimat av tilstandene til systemet. Dette gjøres ved å integrere data fra sensorene. I et forenklet TNS er det tilstander som posisjon, hastighet og orientering som vil være stokastisk optimalt.

Et kalmanfilter krever altså kunnskap om prosess- og måleinstrumentenes dynamikk, den stokastiske beskrivelsen av prosesstøy, målefeil og modellusikkerhet, samt informasjon om initial verdi av tilstandsvariablene. Når denne kunnskapen er tilstede er det mulig å sette opp et kalmanfilter for det aktuelle systemet.

I denne oppgaven er man interessert å se på et diskret Kalmanfilter. Det gis ikke noe detaljert utledning av kalmanfilterlikningene her, men en oppsummering og forklaring slik at man er i stand til å bruke og implementere et kalmanfilter.

Diskret Kalmanfilter

Et Kalmanfilter estimerer en prosess ved hjelp av en tilbakemeldingskontroll: filteret estimerer prosessen en gang, deretter får en tilbakemelding i formen målinger med støy. Likningene for kalmanfilteret faller i to grupper: tidsoppdateringslikninger, og måleoppdateringslikninger. Tidsoppdateringslikningene er ansvarlig for prosjektering fremover i tid på nåværende tilstander for å oppnå a *priori* estimat for neste tids skritt. Mens måleoppdateringslikningene er ansvarlig for tilbakemeldingen for å innlemme en ny måling inn i a *priori* estimat for å få en bedre a *posteriori* estimat. Tidsoppdateringslikningene kan også sees på som prediktor likninger, mens måleoppdateringslikningene kan sees på som korrektur likninger se figur 2.2.



Figur 2.2: Den pågående diskret Kalmanfilter syklus

Kalmanfilterlikningene for det diskrete systemet er som følger[11]:

Tidsoppdatering

$$\underline{\overline{x}}_{k+1} = \phi_k \underline{\hat{x}}_k + \Lambda_k u_k \,, \qquad \underline{\hat{x}}_{0\,gitt} \tag{2.42}$$

$$\overline{P}_{k+1} = \phi_k \hat{P}_k {\phi^T}_k + \Gamma_k Q_k \Gamma^T \,, \hat{p}_{0\,gitt} \tag{2.43}$$

Måleoppdatering

$$K_k = (\overline{P}_k H_k^T \left(H_k \overline{P}_k H_k^T\right) + R_k)^{-1} \tag{2.44}$$

$$\underline{\hat{x}}_k = \underline{\overline{x}}_k + K_k \left(\underline{z}_k - H_k \underline{\overline{x}}_k\right) \tag{2.45}$$

$$\hat{P}_k = (I - K_k H_k)\overline{P}_k \tag{2.46}$$

der

$\underline{\overline{x}}$: a priori estimat av tilstandsvektoren

$\underline{\hat{x}}$: a posteriori estimat av tilstandsvektoren

$\overline{P}$:a priori estimat av tilstandsvektorens kovariansmatrise

$\hat{P}$: a posteriori estimat av tilstandsvektorens kovariansmatrise

K_k: Kalmanfilterforsterkning

Q_k: prosesstøyens kovariansmatrise

R_k: målestøyens kovariansmatrise

2.6 Oppsummering av matematiske modellene

Transformasjonsfeilen referert n-systemet:

Feildefinisjoner		
$\delta \underline{f} = \underline{f}^b - \underline{\tilde{f}}^b$	$\delta \underline{x}^n = \underline{x}^n - \underline{\tilde{x}}^n$	$R_b^n = R(\underline{\varepsilon})\tilde{R}_b^n$
$\delta \underline{\omega} = \underline{\omega}_b^{nb} - \underline{\tilde{\omega}}_b^{nb}$	$\delta \underline{v}^n = \underline{v}^n - \underline{\tilde{v}}^n$	$R(\underline{\varepsilon}) = I + S(\underline{\varepsilon})$

Tabell 2.1: Feildefinisjoner

Fysisk system	Navigasjonslikninger	Feillikninger
$\dot{\underline{x}}^n = \underline{v}^n$	$\dot{\underline{\tilde{x}}}^n = \underline{\tilde{v}}^n$	$\delta \dot{\underline{x}} = \delta \underline{v}$
$\dot{\underline{v}}^n = R_b^n \underline{f}^b - \underline{g}^n$	$\dot{\underline{\tilde{v}}}^n = \tilde{R}_b^n \underline{\tilde{f}}^b - \underline{g}^n$	$\delta \dot{\underline{v}} = -S\left(\tilde{R}_b^n \underline{\tilde{f}}^b\right)\varepsilon + \tilde{R}_b^n \delta \underline{f}$
$\dot{\underline{\theta}} = \mathrm{D}(\theta)\underline{\omega}_b^{nb}$	$\dot{\underline{\tilde{\theta}}} = \mathrm{D}(\theta)\underline{\tilde{w}}_b^{nb}$	$\dot{\varepsilon} = \tilde{R}_b^n \delta \underline{\omega}$

Tabell 2.2: Tre-akset plattform, ikke roterende jord

3 Implementering

Dette kapitelet er delt i to hoveddeler, hvor den første delen tar for seg alt som ble gjort av Java programmering, og applikasjonsutviklingen som gir ut måledata fra sensorene.Den andre delen tar for seg implementeringen av disse måledataene i Matlab, sammen satt av et TNS,og integrasjonsrutinene som ble brukt for å oppnå posisjonen, hastighet, og orienteringen ved bruk av disse målte sensordataene.

3.1 Data logger

Android applikasjoner utvikles ved hjelp av Java. Siden det skal lages en applikasjon i denne oppgaven, er det nødvendig med generell kunnskap om Java-programmering.Som en startfase var det nødvendig å lære hvordan brukersEclipse, laste ned og installere SDK på PC-en og utvikle enkelte Android applikasjoner som for eksempel Hello World applikasjon.

3.1.1 Struktur av en basisAndroid Applikasjon

En Android applikasjon består av fire basis bygningsblokker og disse er[12]:

1. Activity
   En activity er vanligvis en enkel skjerm i et program. Navigering fra en skjerm til en annen gjøres ved å starte en ny aktivitet.

2. Intent and Intent Filters
   Andoird*intent*brukes til å navigere fra en skjerm til en annen. Det består av "action" og "data" der handlingen må være utført. *Intent Filters* gir beskrivelse om hva slags hensikter en aktivitet kan håndtere.

3. IntentReceiver
   IntentReceiver brukes når man vil at koden i programmet skal kjøres som en reaksjon på en ekstern hendelse.

4. Service
   Det er en programkomponent som kjører i bakgrunnen og samhandler ikke med brukeren på en ubestemt tid.

5. Content Provider
   Den blir brukt når et program vil dele sine data som er lagret i filer med et annet program.

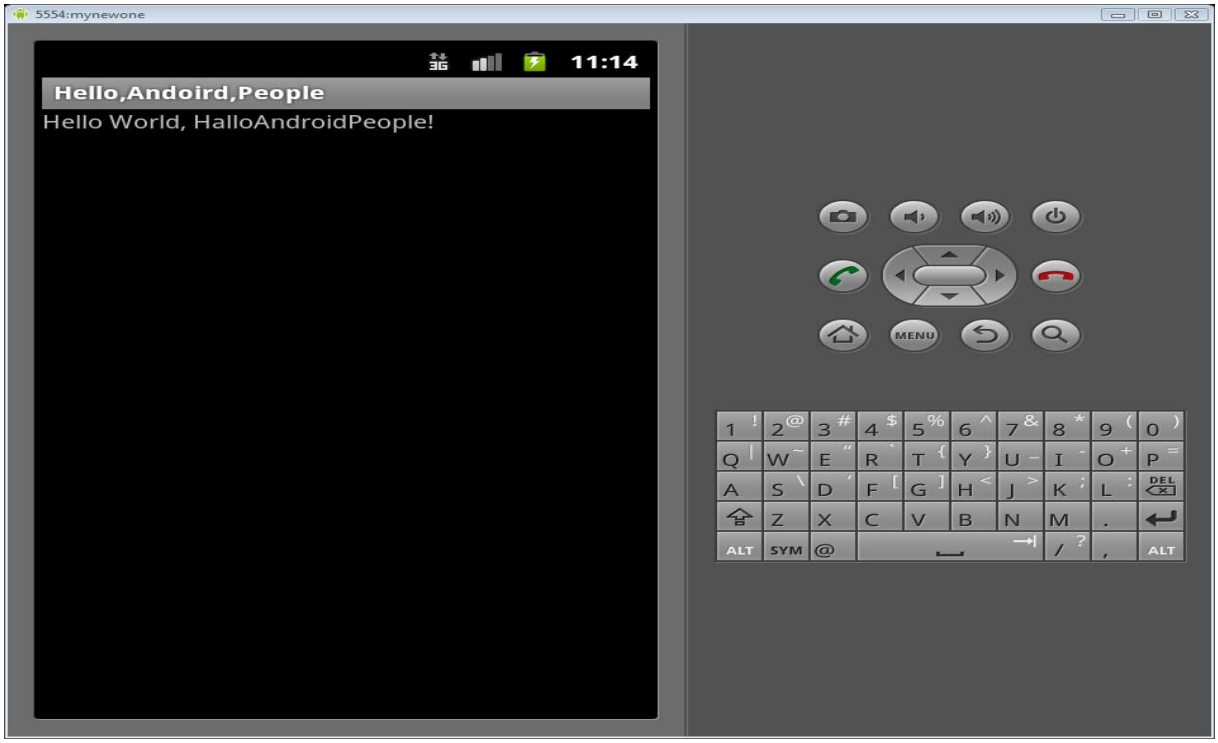
3.1.2 Hello Android applikasjonen

Som en utvikler, vet man at det første inntrykket av et utviklingsmiljø er hvor lett det er å skrive “Hello, World”. På Android er det ganske enkelt også.

Mål

Målet med “Hello, Android, People” applikasjonen er å teste et enkelt program for Android-plattformen. Resultatet av denne leveransen er et selvstendig program som kan ses på emulator skjermen (Fig 3.1). Når brukeren klikker på dette programmet, vises det en melding til brukeren(Fig 3.2).

Figur 3.1:Hello, Andoird, People applikasjon på hovedskjermen på emulatoren



Figur 3.2:Hello, Andoird, People kjøres på emulatoren

Konstruere User Interface(UI)

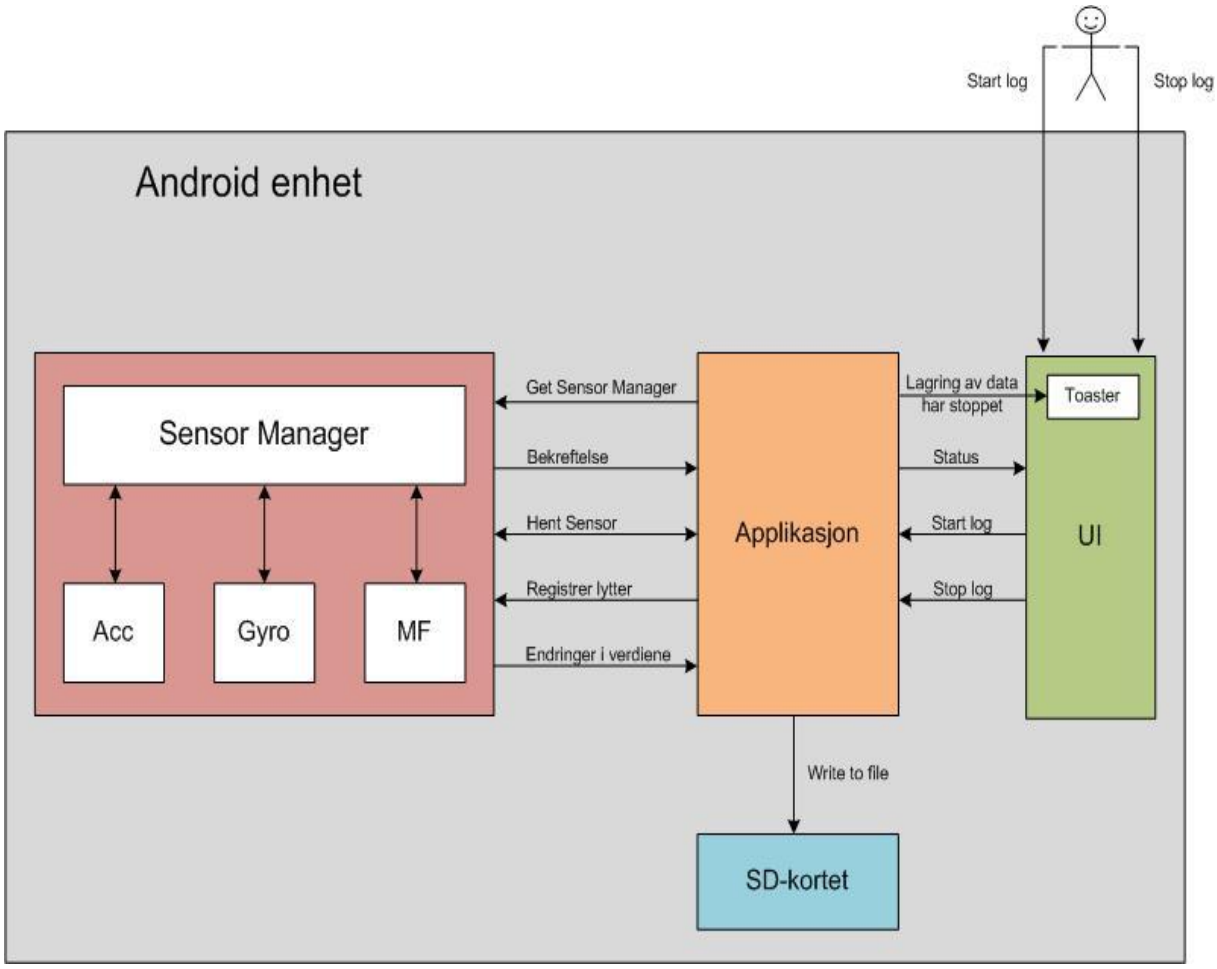
Initialiseringen UI er nødvendig for ådisplayet meldingen til brukeren. dette innebærer å bruke en enkel *TextView* for å vise tilpasset melding. Gitt nedenfor er koden for dette trinnet.

```
TextViewtv = newTextView(this);
/**
    Setter stringverdieravTextView
    */
tv.setText("Hello, Android,People");
/**
    Setter innholdet i activitytilen XML filbaserte layout
    */
setContentView(R.layout.main);
```

Her ble UI konstruert ved å opprette en XML-fil basert layout. I tillegg kan man lage alle UI elementene som tekstbokser, knapper, osv. og referere til hvert av disse elementene ved hjelp av unike ID'erutpeker dem i hoved kildekoden.

3.1.3 Uthenting av sensorsdataene

Målet med applikasjonen*SensorDataRetriever* er å logge dataene fra akselerometer, gyroskop og kompass fra en mobiltelefon, og deretter lagre de dataene i SD kortet som et txt-fil. Hensikten her er å implementere de måledataene i Matlab, og analysere de. Figur 3.3 viser kommunikasjonen mellom de forskjellige komponentene i enheten, og applikasjonen.



Figur 3.3: Kommunikasjonen mellom de forskjellige komponenter

Kommunikasjonen i Androidenheten mellom de forskjellige komponentene og applikasjonen består av følgende trinn:

Get sensor manager: Sensor manager er et program i enheten, det vil si ikke en fysiskenhet. Den kommuniserer direkte med sensorene i Android enheten. Med andre ord for å oppnå kommunikasjon med sensorene, må man be sensor manageren om det.

Bekreftelse: Sensor manageren bekrefter kommunikasjon med sensorene.

Hent sensor: Etter som man har fått bekreftelse fra sensor manageren at sensorene som ble sprut om er tilgjengelige, ber man sensor manageren om å hente de sensorene.

Registrerer lytter: Ved å registrere en lytter på sensoren mottas eventer ved endringer av nøyaktighet eller verdi av sensoren.

Endring i verdiene:Når sensorene endrer nøyaktighet eller verdi, gis beskjed til brukeren.

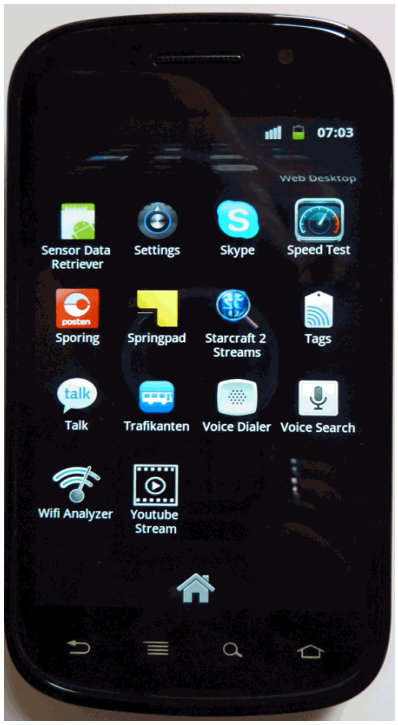
Start log: Brukeren trykker knappen ”Start” i applikasjonen som starter skrivingen til filen.

Stopp log: Brukeren trykker knappen ”Stop” i applikasjonen som avslutter skrivingen til filen.

Status: Applikasjonen kommuniserer med UI’en når en eller flere av sensorene har endret verdier.

Write to file: Skriving til fil skjer kun i perioden etter at brukeren har trykket ”Start” og før han/hun trykker ”Stop”. Skriving starter først når verdier er mottatt fra alle sensorene siden disse ikke mottas samtidig. Når en verdiendring fra en av sensorene mottas skrives denne sammen med de sist mottatte verdiene fra de andre sensorene til filen. Dette er på grunn av at sensorene har ulikt oppdaterings intervall.

Lagring av data har stoppet: Når skrivingen til fil er avsluttet får brukeren beskjed om dette sammen med filnavnet ved hjelp av en Toast (popp opp i Android).



Figur 3.4: Applikasjonen på hovedmenyen til enheten

Figur 3.5: Applikasjonen kjører på enheten uten lagring

Figur 3.6: Applikasjonen kjører på enheten med lagring

Hvis man flytter på telefonen vil tallene endrer seg, med hensyn til x-y og z aksene. Nedenfor er kodene med forklaring på de viktigste trinnene i applikasjonen, vel å merke er en del av applikasjonen unnlatt, grunnen til dette er at trinnene nedenfor er de mest relevante når det gjelder hensikten med denne oppgaven.Hele koden blir lagt til som vedlegg.

Activity: En activity er en enkel, fokusert ting som brukeren kan gjøre. Nesten alle aktiviteter samhandler med brukeren, slik at *activityclass* tar seg av å lage et vindu for oss der man kan plassere vår UI.

```
publicclassSensorDataRetrieverextends Activity implementsSensorEventListener, OnClickListener{
```

Det finnes to metoder nesten alle subklassene i aktiviteten implementerer:

i) onCreateer der man starter aktiviteten. Viktig, her vil man kalle *setContentView(R.layout.main)* med en layout ressurs definerer vår UI.

```
publicvoidonCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main); // skjermensomvisepåtelefonen
```

ii) onPause er der man setter opp hva skal skje når brukeren forlater aktiviteten. Når man forlater applikasjonen fortsetter programmet å kjøre.For å hindre en stor belastning på strømforsyning, det vil si batteriet, bør man ikke logge data fra sensorene mens programmet er inaktivt. Og hvis applikasjon driver å skrive til en fil stopp det også.

```
@Override
protectedvoidonPause()
        super.onPause();
        mSensorManager.unregisterListener(this);// Avregistrer lyttere
        //Stopp skriving til fil hvis dette pågår.
        stopLoggingData();
```

android.hardware.SensorManager gir muligheten til å aksessere enhetens sensorer, det vil si gi informasjon om hvilke sensorer som er tilgjengelig på enheten, samt å registrere lyttere på spesifikke sensorer som vil gi meldinger når en sensor endrer verdi, nøyaktighet og lignende.

```
// SensorManagerenhentesfraContext.getSystemService()
sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
```

Avhengig av hvilken enhet som applikasjonen kjøres på kan flere ulike sensorer være tilgjengelige. Ikke bare av ulike typer som akselerometer, gyroskop også videre, men også ulike sensorer av samme type. Dersom en enhet har en eller flere sensorer av samme type vil alltid en av disse være ”standard” sensoren for den spesifikke typen. Standard sensoren vil som regel være den som passer til de fleste formål, det vil si den beste kombinasjonen av oppløsning, nøyaktighet og oppdaterings frekvens.

```
// Bruker SensorManageren til å hente standard akselerometer
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Ved å registrere en lytter på sensoren mottas *eventer* ved endringer av sensoren. Eventene mottas ved at metodene *onAccuracyChanged* og *onSensorChanged* fra *SensorEventListener*interfacet kalles. Dette betyr at klassen som registreres som lytter må implementere dette interfacet.

Når en lytter registreres spesifiseres også hvor ofte endringer på sensoren skal sendes til lytteren. Sensoren vil gi meldinger til systemet så fort den kan, men systemet begrenser hvor ofte den registrerte lytteren skal få beskjed om endringer av sensoren. Denne spesifikasjonen må ses på i sammenheng med hva dataen skal brukes til. Hvis sensor dataen bare brukes til å oppdatere brukergrensesnittet vil en lav oppdaterings hastighet være ønskelig, siden det krever en del resurser å oppdatere skjerm bildet.

```
// Registrerer en lytter på denne klassen med en høy oppdaterings frekvens.

sensorManager.registerListener(this, accelerometer,

SensorManager.SENSOR_DELAY_FAST);
```

Metoden*onAccuracyChanged*er en del av*SensorEventListener*interfacet. Denne metoden kalles ved endring i nøyaktigheten til sensoren og er relevant for posisjons sensorer (GPS), der nøyaktigheten kan være svært varierende. For akselerometer og gyroskop sensorer vil nøyaktigheten stort sett være konstant høy.

```
@Override
publicvoidonAccuracyChanged(Sensor sensor, int accuracy) {

	//Akselereometeret finnes og har endret nøyaktighet
	if (accelerometer != null&& sensor == accelerometer)
	{
		switch(accuracy)
		{
		caseSensorManager.SENSOR_STATUS_UNRELIABLE:
			// Unreliable
			break;
		caseSensorManager.SENSOR_STATUS_ACCURACY_LOW:
			// Low
			break;
		caseSensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
			// Medium
			break;
		caseSensorManager.SENSOR_STATUS_ACCURACY_HIGH:
			// High
			break;
		default:
			//Unknown
		}
	}
```

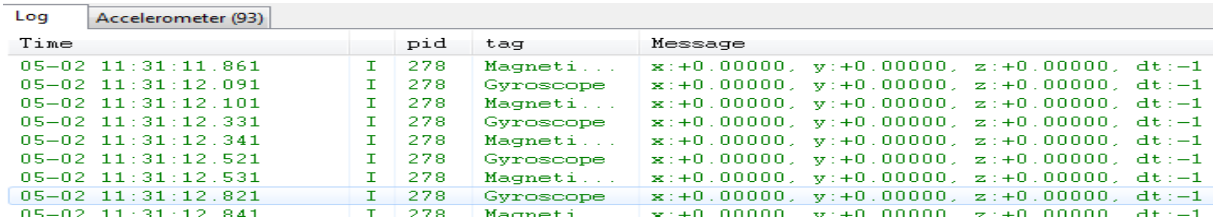
Metoden*onSensorChanged* er en del av*SensorEventListener*interfacet. Denne metoden kalles ved verdi endring i dataen fra sensoren.

```
@Override
publicvoidonSensorChanged(SensorEvent event) {

	//Akselereometeret finnes og har endrede verdier
	if(accelerometer != null&&event.sensor == accelerometer)
	{
		longtimeStamp = event.timestamp; // Tiden i nano sekunder
		float x = event.values[SensorManager.DATA_X]; // Akselerasjon i x
		float y = event.values[SensorManager.DATA_Y]; // Akselerasjon i y
		float z = event.values[SensorManager.DATA_Z]; // Akselerasjon i z
	}
```

Logging av data til fil:

Det finnes en egen klasse i Android for å sende meldinger til en ’applikasjons log’ som Eclipse kan lese. Dette var forsøkt først for å lagre dataene der ifra, men lasten på enheten ble for høy, og ikke alle tekstene kom. For å illustrere hvordan dette ser ut, eksempel på N antall målinger fra gyroskop- og kompas sensorene vises under.



Log | Accelerometer (93)

Time		pid	tag	Message
05-02 11:31:11.861	I	278	Magneti...	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.091	I	278	Gyroscope	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.101	I	278	Magneti...	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.331	I	278	Gyroscope	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.341	I	278	Magneti...	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.521	I	278	Gyroscope	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.531	I	278	Magneti...	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.821	I	278	Gyroscope	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1
05-02 11:31:12.841	I	278	Magneti	x:+0.00000, y:+0.00000, z:+0.00000, dt:-1

Tabell 3.1: DDMS, måledata fra Akselerometer, gyroskop- og kompass

Siden meldinger til applikasjons loggen ikke kunne gjøres, ble en egen klasse for logging til en fil på SD-kortet utviklet i stede.

Start logging

Klassen initialiseres ved å gi ønsket filnavn og mappenavn. Filnavnet som brukes er Log_ dagens dato_klokkeslettet dataene ble hentet i. For eksempel dataene som blir hentet den 03/06/2011, kl 15:08:37 har fil navnet: Log_20110603_150837.

```
logWriter= newDataWriter("Log_"+Helpers.TimeNow(Helpers.FILE_DATE_FORMAT_NOW),
"SensorDataRetriver");
```

Mappenavnet bestemmes i klassen*public DataWriter*. Hvis mappenavnet ikke er spesifisert, brukes det roten til SD kortet, men hvis den eksisterer fra før bruk det samme navnet.

```
publicDataWriter(String fileName, String dirName) throwsIOException,
       {

              this.FileName = fileName;


              File dir;

              if(dirName.isEmpty())
              {
       //Hvis directory navnet er ikke spesifisert bruker vi roten til SD kortet
              dir = Environment.getExternalStorageDirectory();
              }
              else
              {
       //Hvis directory navnet eksisterer, bruk det
              dir = new
File(Environment.getExternalStorageDirectory().getAbsolutePath() +'/'+
dirName);

File file = new File(dir, fileName);
file.createNewFile();
```

Eventer sendes til objekte ved å bruke metoden *Write(SensorEventevent)* som oversettes i klassen til *Write(int SensorType, long timeStamp, float x, float y, float z)*.

```
	publicvoid Write(intSensorType, longtimeStamp, float x, float y,
float z)
	{
		switch(SensorType)
		{
			caseSensor.TYPE_ACCELEROMETER:
				accelData.SetValues(x, y, z);
				break;
			caseSensor.TYPE_GYROSCOPE:
				gyroData.SetValues(x, y, z);
				break;
			caseSensor.TYPE_MAGNETIC_FIELD:
				magData.SetValues(x, y, z);
				break;
			default:
				return; // Ignorer altannet
		}

		this.timeStamp = timeStamp;
		writeDataToFile();
	}
```

I metodenwriteDataToFile()

```
privatevoidwriteDataToFile()
	{

if(accelData.HasValues() &&gyroData.HasValues() &&magData.HasValues())
		{
```

Det som menes med koden overfor erHasValues er ture dersom verdier er mottatt en gang.

```
	privatevoidprintData(SensorData data, charseperator)
	{
		//Print<data.x><seperator><data.y><seperator><data.z>
		writer.print(data.x);
		writer.print(seperator);
		writer.print(data.y);
		writer.print(seperator);
		writer.print(data.z);
	}
```

Hvis HasValues er true (verdier er mottatt en gang) skriv ut de verdiene til fil med en separator som utvikleren bestemmer selv. I denne oppgaven komma separator har blitt brukt. Grunnen til dette er Matlab forstår komma separasjon.

Utvikleren kan enkelt bytte separator, ved å bytte (“,”) med en annen separator etter ønske.

```
charseperator = ';';
```

stopp logging

Ved å kalle metoden Dispose() stoppes skriving til fil.

```
publicvoid Dispose()
{
        isDisposed = true;
        writer.close();
}
```

Som ble nevnt tidligere måle verdier fra sensorene blir lagret i SD-korte som en txt-fil format. Dette filet blir implementert i Matlab. Dataene består av N antall kolonner, hvor hver av disse presenterer noe vi bestemmer i *SensorDataRetriever* applikasjonen. I dette tilfelle består filet av 10 kolonner hvor kolonne 1 representerer tid i nanosekunder, kolonne 2-4 presenterer målt akselerometer verdiene i x,y og z aksene, kolonne 5-7 presenterer målt gyroskop verdiene og 8-10 presenterer målt kompass verdiene. Etter som det er veldig store unøyaktigheter på verdiene fra kompass sensoren, ble det bestemt å ikke bruke de.

For å illustrere hvordan filet ser ut se tabellen under (tabell 3.2).

```
3700878828000,-2.2026656,4.6351743,8.293514,-3.943746,-0.37140608,1.001819,-4.375,-41.3125,11.75
3700887056000,-2.2026656,4.6351743,8.293514,-3.813021,-0.64996064,0.93218035,-4.375,-41.3125,11.75
3700896293000,-2.2026656,4.6351743,8.293514,-3.813021,-0.64996064,0.93218035,-5.875,-39.8125,9.75
3700896567000,-2.2026656,4.6351743,8.293514,-3.6004398,-0.9932669,0.90285885,-5.875,-39.8125,9.75
3700898862000,-2.2218192,4.711789,9.155427,-3.6004398,-0.9932669,0.90285885,-5.875,-39.8125,9.75
3700906100000,-2.2218192,4.711789,9.155427,-3.4281757,-1.2412782,0.9211848,-5.875,-39.8125,9.75
3700915649000,-2.2218192,4.711789,9.155427,-3.3646457,-1.282817,0.95294976,-5.875,-39.8125,9.75
3700916824000,-2.2218192,4.711789,9.155427,-3.3646457,-1.282817,0.95294976,-7.125,-38.0625,8.25
3700918895000,-2.0494366,3.9456444,7.3549876,-3.3646457,-1.282817,0.95294976,-7.125,-38.0625,8.25
3700925188000,-2.0494366,3.9456444,7.3549876,-3.3744197,-1.1948525,0.9578367,-7.125,-38.0625,8.25
3700934694000,-2.0494366,3.9456444,7.3549876,-3.3988543,-1.0238101,0.9627236,-7.125,-38.0625,8.25
3700936816000,-2.0494366,3.9456444,7.3549876,-3.3988543,-1.0238101,0.9627236,-8.125,-35.0625,7.25
3700939463000,-2.0685902,3.2944214,7.6614456,-3.3988543,-1.0238101,0.9627236,-8.125,-35.0625,7.25
3700944513000,-2.0685902,3.2944214,7.6614456,-3.385415,-0.80756384,1.0103711,-8.125,-35.0625,7.25
3700953805000,-2.0685902,3.2944214,7.6614456,-3.3377676,-0.6059783,1.1472049,-8.125,-35.0625,7.25
3700957530000,-2.0685902,3.2944214,7.6614456,-3.3377676,-0.6059783,1.1472049,-9.375,-32.0625,5.5
3700958947000,-2.1260512,3.3135753,9.11712,-3.3377676,-0.6059783,1.1472049,-9.375,-32.0625,5.5
```

Tabell 3.2: Målt sensor data (txt–fil)

3.2 Treghetsnavigasjonssystem

Som sagt tidligere inneholder håndholdte enheten ulike sensorer som kan brukes i et treghetsnavigasjonssystem. Ved å måle vinkelhastighet og spesifikk kraftkan man integrere dette opp til orientering, fart og posisjon. Det antas at aksene til sensorene er parallelle med legemerammen. Denne rammen ligger fast i mobilen, med origo i massesenteret. Som det ble nevnt tidligere blir målingene påvirket av støy. For å unngå store unøyaktigheter bruker man matematiske modeller, slik at man kan korrigere målingene, ved bruk av et kalmanfilter.

Det har blitt forklart og satt opp i kapitel 2, navigasjonslikninger, feilmodellikninger, og kalmanfilterlikningene. Pseudokodene for Navigasjonslikningene, feilmodellikningene, og kalmanfilter likningene blir implementert her:

3.2.1 Pseudokode for navigasjonslikningene

Nå har vi alle likningene som trengs for å sette opp pseudokode for navigasjonslikningene, siden logging av data skjer i diskret tid(t), bruker vi Heun's metode for løsning av navigasjonslikningene:

Program:

$t_k, \underline{\tilde{f}}^b(t_k)\ og\ \underline{\tilde{\omega}}_b^{nb}(t_k)$ er gitt for k=0:N

$$\underline{x}_0 = \begin{bmatrix} p_b^n(t_0) \\ v_b^n(t_0) \\ \theta(t_0) \end{bmatrix}$$

for k = 0 : N-1

h = t_{k+1} - t_k

$$u_k = \begin{bmatrix} \underline{\tilde{f}}^b(t_k) \\ \underline{\tilde{\omega}}_b^{nb}(t_k) \end{bmatrix} \qquad u_{k+1} = \begin{bmatrix} \underline{\tilde{f}}^b(t_{k+1}) \\ \underline{\tilde{\omega}}_b^{nb}(t_{k+1}) \end{bmatrix}$$

$$x'_{k+1} = x_k + h * f(\underline{x}_k, \underline{u}_k, \underline{t}_k)$$

$$x_{k+1} = x_k + \frac{h}{2}\left[f(\underline{x}_k, \underline{u}_k, \underline{t}_k) + f(\underline{x'}_{k+1}, \underline{u}_{k+1}, \underline{t}_{k+1})\right]$$

end

Funksjon:= $f(\underline{x}, \underline{u}, \underline{t})$

$$p_b^n = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad v_b^n = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix}, \quad = \begin{bmatrix} x_7 \\ x_8 \\ x_9 \end{bmatrix} \quad g = 9.81\, m/s^2$$

$$\underline{\tilde{f}}^b = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \underline{\tilde{\omega}}_b^{nb} = \begin{bmatrix} u_4 \\ u_5 \\ u_6 \end{bmatrix} \quad g^n = [0; 0; -g]$$

$\tilde{R}_b^n = R_b^n(\underline{\theta})$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = v_b^n, \quad \begin{bmatrix} f_4 \\ f_5 \\ f_6 \end{bmatrix} = \tilde{R}_b^n \underline{\tilde{f}}^b - \underline{g}^n, \quad \begin{bmatrix} f_7 \\ f_8 \\ f_9 \end{bmatrix} = \mathrm{D}(\theta)\underline{\tilde{\omega}}_b^{nb}$$

Matlab koden for implementering av sensor data står under, hele Matlab koden ble lagt til som vedlegg.

```
a=importdata('logg25.mat');

[N,Ns]=size(a);


t0=a(1,1);

ta=(a(:,1)-a(1,1))*(10^-9);
f__b = [a(:,2)'; a(:,3)'; a(:,4)']; % Spesfikkkraft

w_b_nb = [a(:,5)'; a(:,6)'; a(:,7)']; %Vinkelhastighet
```

3.2.2 Pseudokode for feilmodelllikningene

Nå har vi alle likningene som trengs for å sette opp pseudokode for feilmodellen:

Program:

$\delta f = \underline{f}^b - \underline{\tilde{f}}^b \Rightarrow \underline{\tilde{f}}^b = \underline{f}^b + \beta^a + v^a,\ \underline{f}^b = 0$

$\delta\omega = \underline{\omega}_b^{nb} - \underline{\tilde{\omega}}_b^{nb} \Rightarrow \underline{\tilde{\omega}}_b^{nb} = \underline{\omega}_b^{nb} + \beta^g + v^g, \underline{\omega}_b^{nb} = 0$

Setter inn i likningene for navigasjonssystem, får vi:

$\underline{\hat{x}}_0 = 0, \hat{p}_0 = 0$ (initial verdiene)

$$\delta\dot{p}^n = \delta v^n$$

$$\delta\dot{v}^n = -\mathrm{S}(\tilde{R}_b^n \underline{\tilde{f}}^b)\epsilon^n - \tilde{R}_b^n \beta^a - \tilde{R}_b^n v^a$$

$$\dot{\varepsilon}^n = -\tilde{R}_b^n \beta^g - \tilde{R}_b^n v^g$$

$$\dot{\beta}^a = 0$$

$$\dot{\beta}^g = 0$$

$H = [0\ 1\ 0\ 0\ 0]$

$\delta\underline{\dot{x}} = F(\underline{\tilde{x}}) + G\underline{v}$

$$\begin{bmatrix} \delta\dot{p}^n \\ \delta\dot{v}^n \\ \dot{\varepsilon}^n \\ \dot{\beta}^a \\ \dot{\beta}^g \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\mathrm{S}(\tilde{R}_b^n \underline{\tilde{f}}^b) & -\tilde{R}_b^n & 0 \\ 0 & 0 & 0 & 0 & -\tilde{R}_b^n \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta p^n \\ \delta v^n \\ \epsilon^n \\ \beta^a \\ \beta^g \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\tilde{R}_b^n & 0 \\ 0 & -\tilde{R}_b^n \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v^a \\ v^g \end{bmatrix}$$

$$\delta\underline{z_k} = H(\underline{x_k}) + \underline{w_k}$$

3.2.3 Pseudokode for kalmanfilterlikningene

I denne oppgaven brukes det diskret kalmanfilter, likningene til det diskret kalmanfilteret følger av oversikt tabellen gitt i [11]:

$$\delta\dot{\underline{x}} = F * \delta(\underline{x}, t) + G * \underline{v} \tag{3.1}$$

$$\delta\underline{z_k} = H * (\underline{x_k}) + \underline{w_k} \tag{3.2}$$

Pseudokode for kalmanfilterlikningene

Nå har vi alle likningene som trengs for å sette opp pseudokode for kalmanfilteret:

Kalmanfilter:

$\underline{\hat{x}}_0 = 0, \hat{p}_0 = 0$ (initial verdiene)

$$H = [0\ 1\ 0\ 0\ 0]$$

$R = 0.0$

$Q = 0.0$

// R, og Q blir satt til 0 siden vi ikke vet hva er sann verdiene for dem.

For k = 1:N

$$\underline{\overline{x}}_{k+1} = \varphi_k \underline{\hat{x}}_k$$

$$\overline{P}_{k+1} = \varphi_k \hat{P}_k {\varphi^T}_k + \Gamma_k Q_k \Gamma^T, \hat{P}_{0\ gitt}$$

$$K_k = (\overline{P}_k H_k^T (H_k \overline{P}_k H_k^T) + R_k)^{-1}$$

$$\underline{\hat{x}}_k = \underline{\overline{x}}_k + K_k \left(\underline{z}_k - H_k \underline{\overline{x}}_k\right)$$

$$\hat{P}_k = (I - K_k H_k)\overline{P}_k$$

end

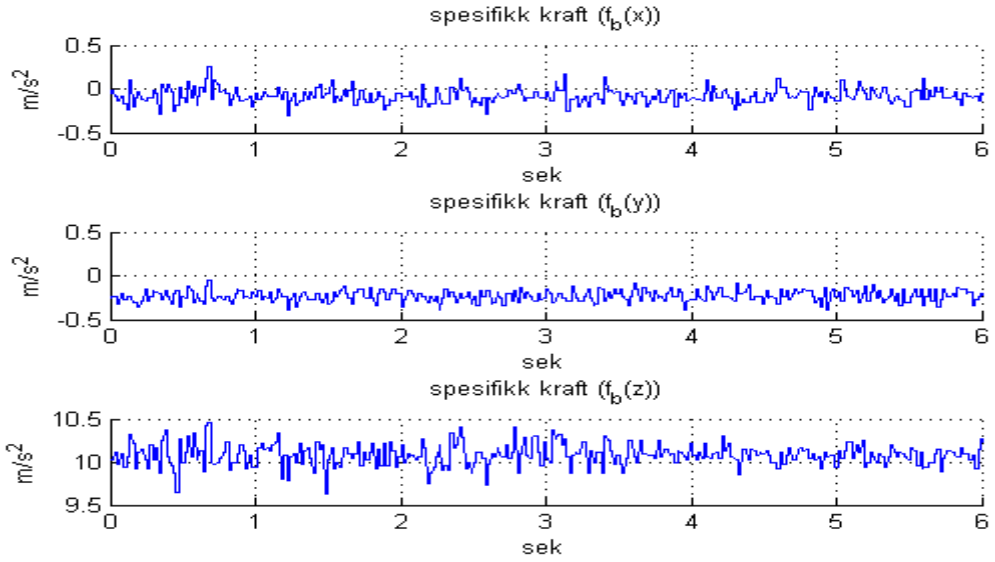
4 Resultater og diskusjon

I denne oppgaven ble det utviklet en applikasjon i Java som logger data fra forskjellige sensorer, og lagrer de dataene i SD-kortet på mobiltelefonen. Disse måledataene, som ble nevnt tidligere implementeres i Matlab hvor treghetsnavigasjonssystem ble satt sammen. For å analysere dette, ble 3 forskjellige sett av data tatt med. Første sett inneholder 1480 målinger fra sensorene, der mobiltelefonen står i ro i 6 sekunder uten rotasjon.Andre sett inneholder 1960 målinger, der mobiltelefonen står i ro i 9 sekunder og roteres ca. 90 grader. Det sistesettet inneholder 4299 målinger. I dette tilfellet beveges mobiltelefonenkvadratisk med en avstand på 25 cm. Ved alle de fire kantene står mobiltelefonen i ro i ca. 2 sekunder, som gir et totalt tidsintervall på 20 sekunder.

Plottingav akselerometersdataene (spesifikk kraft), gyroskopsdataene (vinkelhastighet), posisjonen, hastigheten, og orienteringen i alle tre x, y og z aksene vises for alle settene.

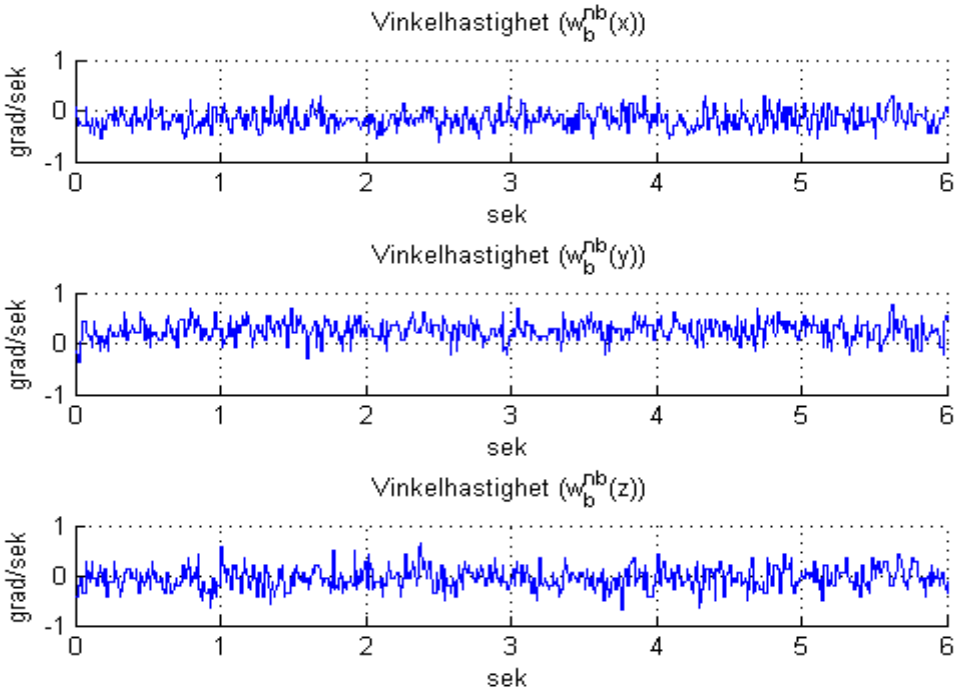
4.1 Sett 1: Enhetenstår i ro uten rotasjoner

Figur 4.1.1, og 4.1.2 viser rå målinger fra akselerometrene og gyroskopene, mens figur 4.1.3, 4.1.4 og 4.1.5 viser posisjonen, hastigheten orienteringen som er beregnet ved bruk av de rå dataene.



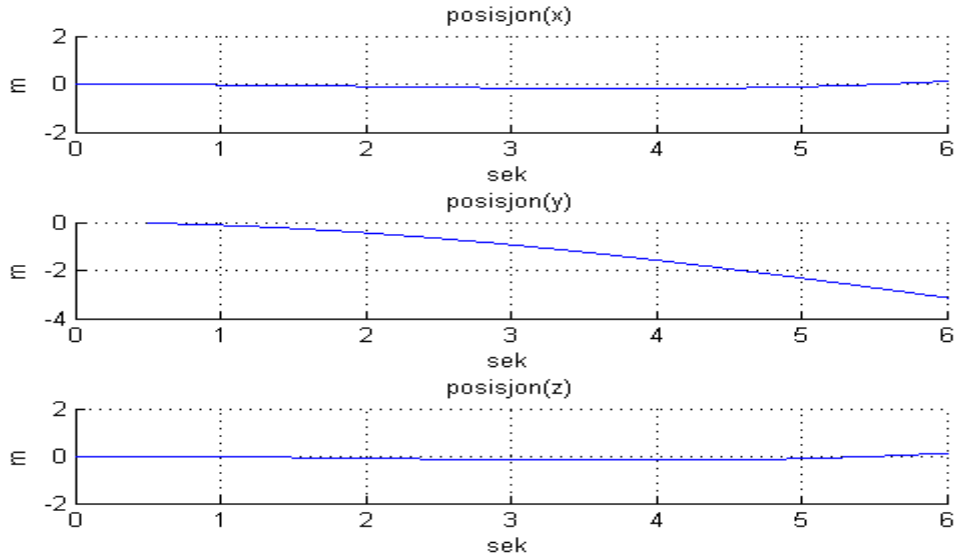
Figur 4.1.1: Målt spesifikk kraft, mobiltelefonen står i ro, og roteres ikke.

Ut fra Figur 4.1.1 ser vi at den spesifikke kraften blir påvirket av bias feil, og andre typer støy. Den holder seg stabilt rundt null hele tiden, bortsett fra ved z-aksen der den spesifikke kraften stabiliserer seg rundt 10 på grunn av gravitasjonskreftene.



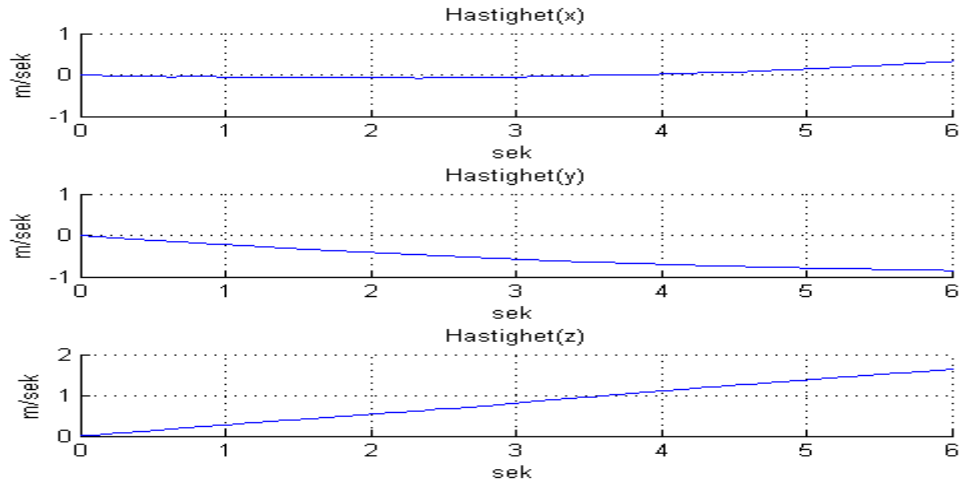
Figur 4.1.2: Målt vinkelhastighet, mobilen står i ro og roteres ikke.

Figur 4.1.2 viser vinkelhastigheten når mobilen er i ro og ved ingen rotasjon. Vinkelhastigheten holder seg rundt 0 ved alle tre aksene siden det ikke er noe rotasjon, mensden blir påvirket av støy og andre feilkilder fra gyroskopsmålingene.



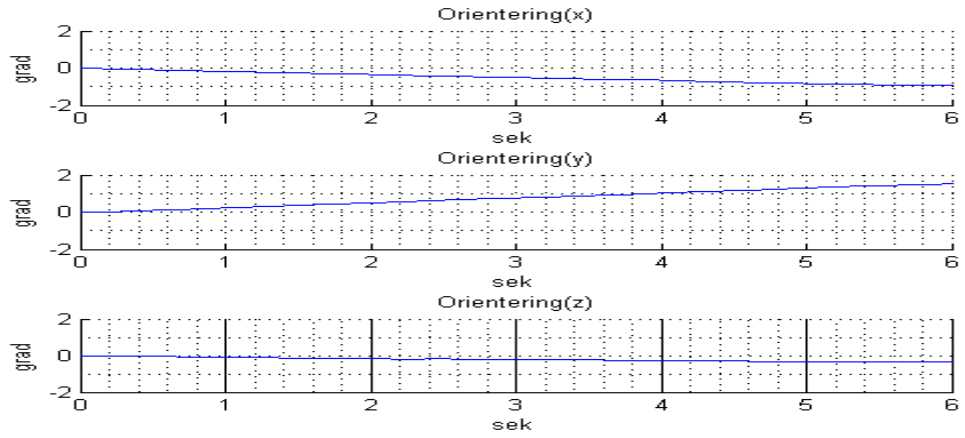
Figur 4.1.3: Beregnet posisjon, ved bruk av navigasjonslikningene

Ut ifra figur 4.1.3 ser vi at posisjonen ved x – og z aksen er stabil ved 0, ettersom mobilen ligger i ro. Vi ser videre at verdiene ved y-aksen avviker veldig fra 0. Årsaken til dette kan være integrasjon av bias feilen eller at mobilen står på skakk. Merk at den spesifikke kraften ved y-aksen har betraktelig mer bias feil en ved de andre aksene, noe som fører til at posisjonen ved y-aksen variere mer enn posisjonen ved de to andre aksene.



Figur 4.1.4: Beregnet hastighet, ved bruk av navigasjonslikningene

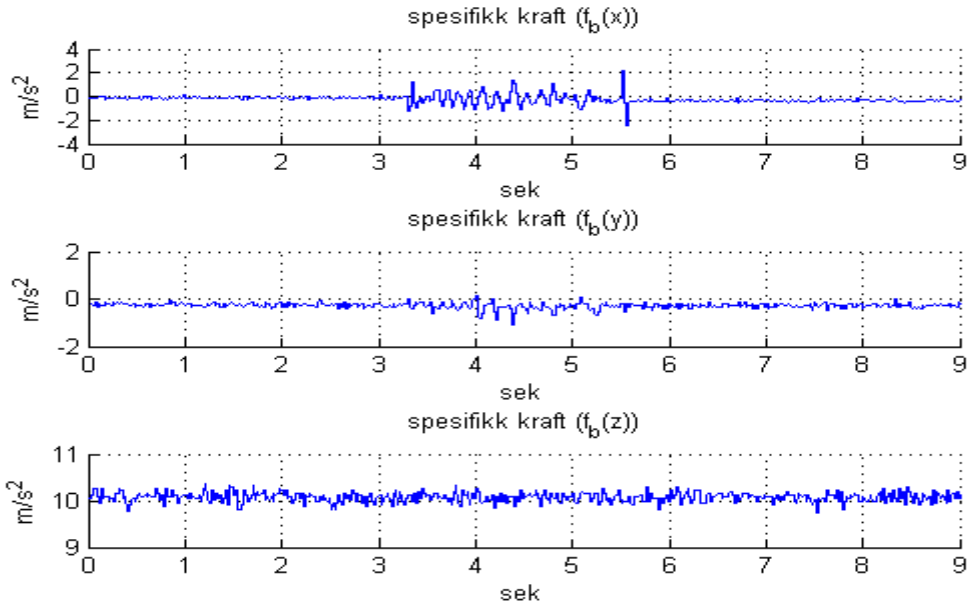
Figur 4.1.4 og 4.1.5 presenterer henholdsvis hastigheten og orienteringen. Begge figurene viser minimal endring siden mobilen står i ro, og det ikke er noe rotasjon ved noen av aksene. Avviket fra 0 er grunnet støyog andre feilkilder i signalet.



Figur 4.1.5: Beregnet orientering, ved bruk av navigasjonslikningene

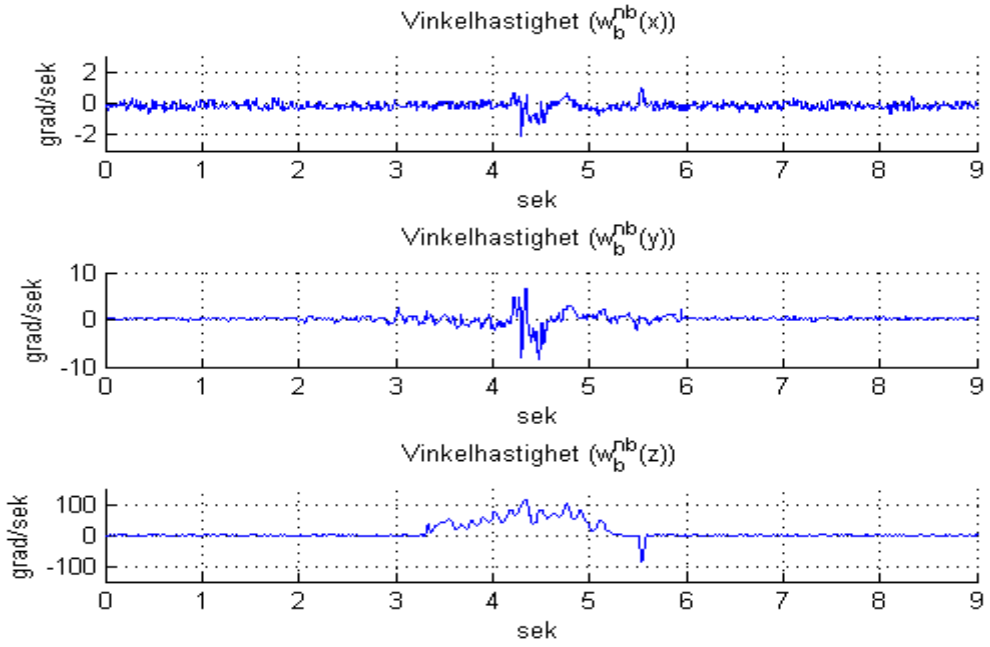
4.2 Sett 2: Enheten står i ro med 90 grader rotasjon

Figur 4.2.1 viser endringen i den spesifikke kraften ved rotasjon på 90 grader. Ettersom det ikke er noe avstands bevegelse av mobilen ser vi at aksene holder seg relativt stabile rundt 0, bortsett fra z-aksen som blir påvirket av gravitasjonskreftene. I tidsintervallet mellom 3 og 6 sekunder skjer vi litt variasjon i x-aksen. Det er da rotasjonen av mobilen skjer, og siden mobilen roteres manuelt blir akselerometer signalet påvirket.



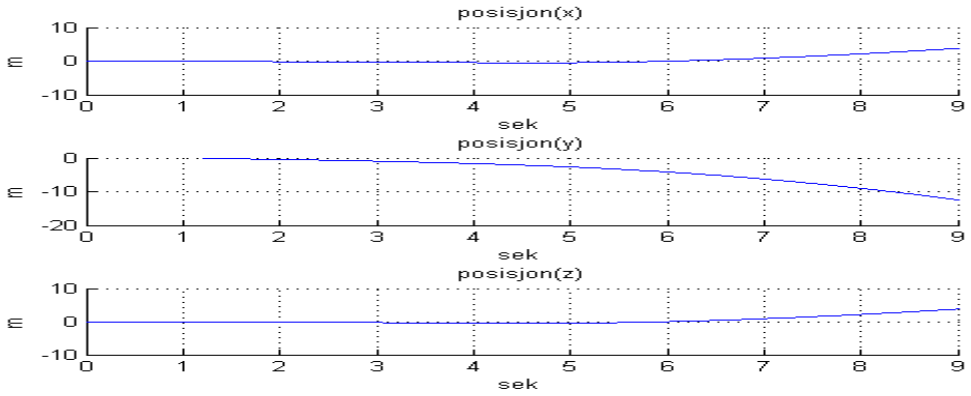
Figur 4.2.1: Målt spesifikk kraft, mobiltelefonen står i ro og roteres 90 grader.

Ut ifra figur 4.2.2 ser vi endringen i gyroskop signalet ved en horisontal rotasjon på 90 grader. Ettersom rotasjonen skjer om z-aksen i tidsintervallet mellom 3-6 sekunder, ser vi at vinkelhastigheten i denne perioden øker til ca. 100 grader og minker raskt til 0 igjen. Siden rotasjonen er utført manuelt, ble ikke vinkelen målt nøyaktig til 90 grader men til ca 100 grader. Manuell rotasjon er også årsaken til litt avvik ved x-og y-aksen.



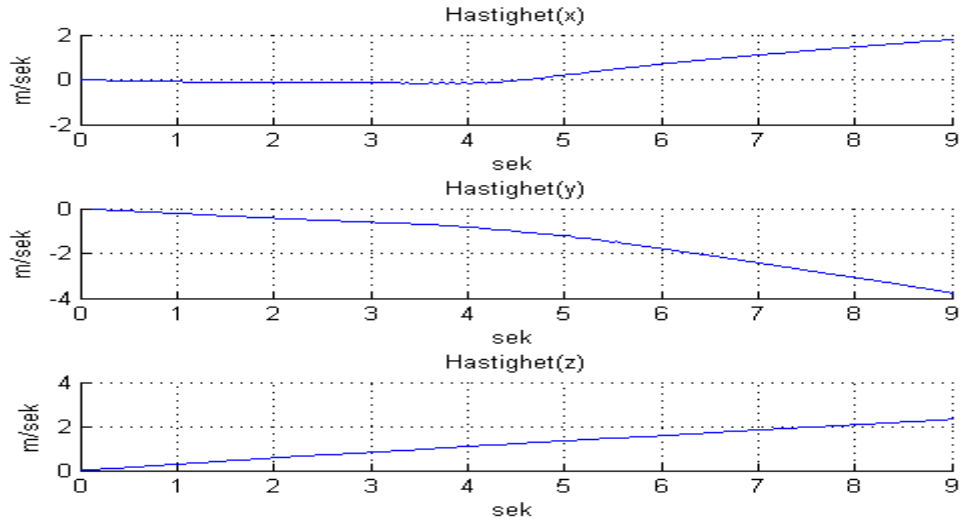
Figur 4.2.2: Målt vinkelhastighet, mobilen står i ro og roteres 90 grader.

Figur 4.2.3 viser endringen i posisjonen ved en horisontal rotasjon på 90 grader. X-aksen og z-aksen opplever ingen nevneverdig avvik ettersom det ikke er noe stor bevegelse av mobilen i disse retningene. Vi ser derimot at y-aksen begynner å minke etter en stund. Dette er grunnet påvirkningen av bias feilen, og andre typer støy.



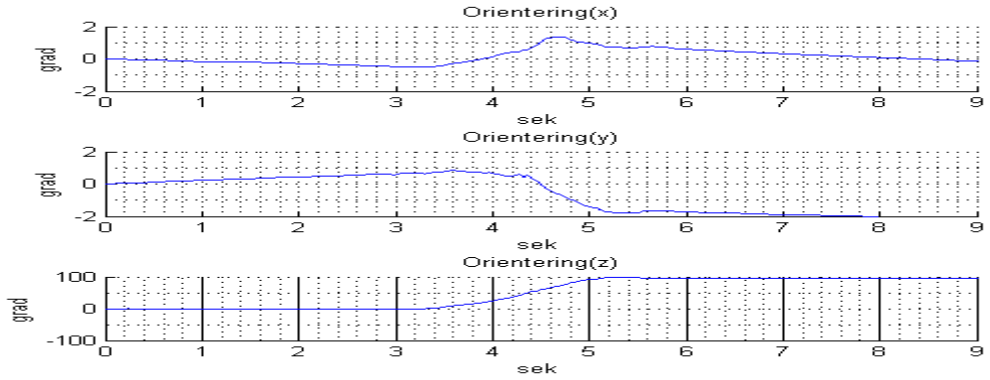
Figur 4.2.3: Beregnet posisjon, ved bruk av navigasjonslikningene

Fra figur 4.2.1 har vi sett at de rå måledataene fra akselerometeret ble påvirket av rotasjonene til mobilen. Figur 4.2.4 viser de beregnede hastighetene basert på de samme måledataene som fra figur 4.2.1. Siden vi bruker en integrasjonsrutine for beregning av hastigheten, ser vi at avviket integreres opp og øker. For å kunne unngå dette avviket er det anbefalt å bruke et kalmanfilter.



Figur 4.2.4: Beregnet hastighet, ved bruk av navigasjonslikningene

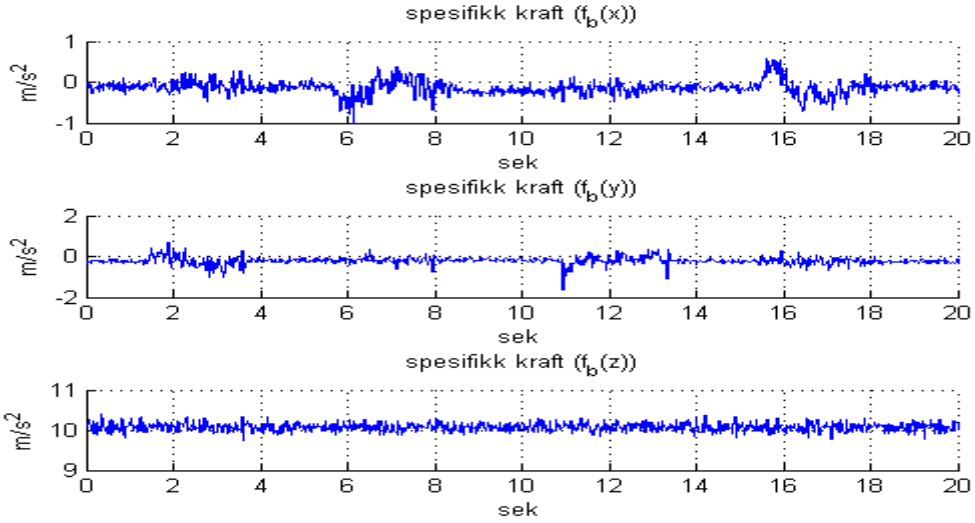
Figur 4.2.5 viser rotasjonen om z-aksen beregnet ved bruk av gyroskop signalet. X-og y-aksen blir som vi ser påvirket på grunn av manuell rotasjon og gir et lite avvik. Z-aksen starter på null, og stiger opp til 90-100 grader som følge av rotasjonen.



Figur 4.2.5: Beregnet orientering, ved bruk av navigasjonslikningene

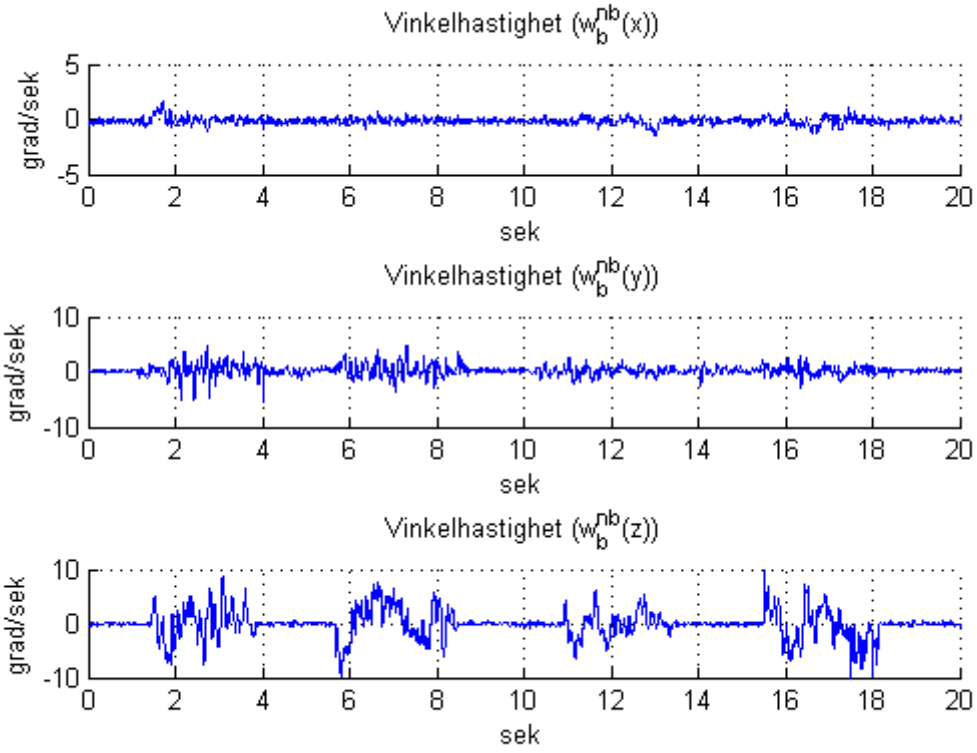
4.3 Sett 3: Enheten beveges kvadratisk

Figur 4.3.1 viser den målte spesifikke kraften (rå data fra akselerometeret) når mobilen er i bevegelse. Når mobilen er i ro, ser vi at det ikke er noe variajon i grafene. Derimot når mobilen er i bevegelse så får vi et utslag ved disse tidsintervallene. Denne variasjonen synes mye bedre fra gyroskop signalet. Grunnen til dette kan være følsomheten til sensorene. På grunn av gravitasjonskreftene er verdien ved z-aksen 10 i stedet for 0.



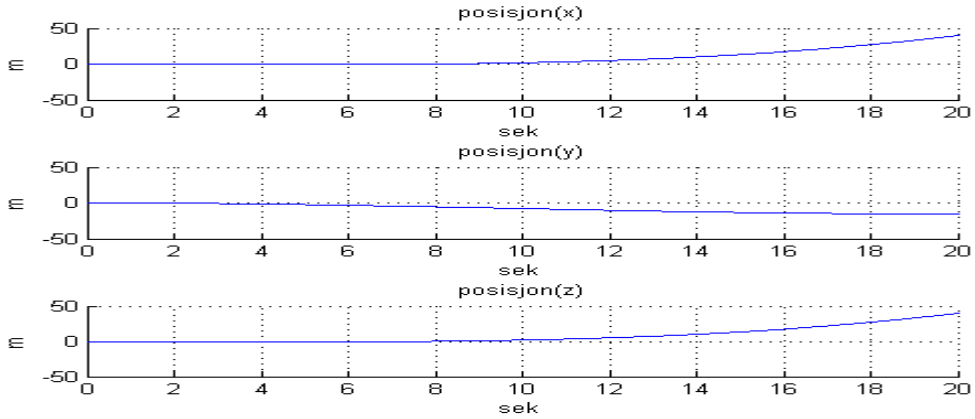
Figur 4.3.1: Målt spesifikk kraft, mobiltelefonen er i bevegelse.

Fra figur 4.3.2 ser vi målte vinkelhastigheter (rådata fra gyroskopet) ved en firkantbevegelse av mobiltelefonen. I selve bevegelsen er det ingen rotasjon. De tidsintervallene der mobilen er i bevegelse vises veldig klart i denne figuren. Når mobilen er i ro, er vinkelhastighetene tilnærmet like 0.

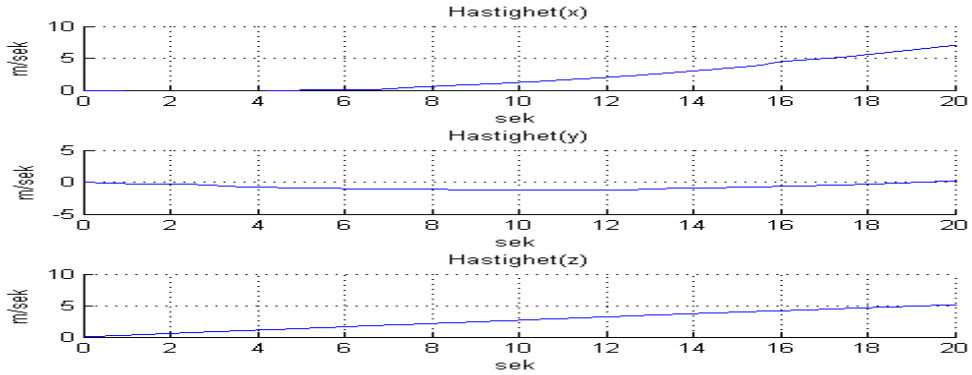


Figur 4.3.2: Målt vinkelhastighet, mobiltelefonen er i bevegelse.

Figur 4.3.3 og Figur 4.3.4 viser henholdsvis den beregnede posisjonen, og den beregnede hastigheten ved bruk av akselerometer signalet, og integrasjonsrutiner. På grunn av støy og andre feilkilder fra signalet blir ikke resultatene som ønsket . Derfor må man bruke et kalmanfilter for å begrense påvirkningen av disse feilkildene.

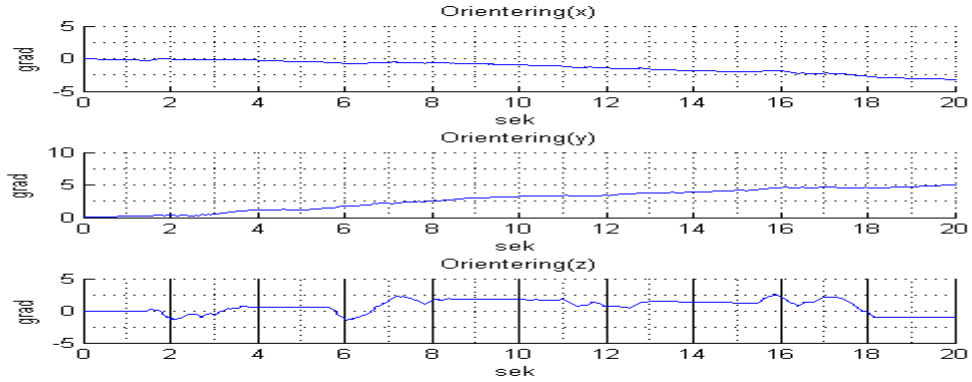


Figur 4.3.3: Beregnet posisjon, ved bruk av navigasjonslikningene



Figur 4.3.4: Beregnet hastighet, ved bruk av navigasjonslikningene

Figur 4.3.5 viser den beregnede orienteringen ved bruk av gyroskop signalet. På grunn av ingen rotasjon av mobilen, vil ideelt sett alle de tre aksene være tilnærmet lik 0. Ettersom bevegelsen skjer manuelt, må man alltid regne med litt variasjon i vinklene.



Figur 4.3.5: Beregnet orientering, ved bruk av navigasjonslikningene

5 Konklusjonog videre arbeid

5.1 Konklusjon

Android applikasjoner utvikles vanligvis i Java ved å bruke Androids SDK. For å realisere Android applikasjonen måtte jeg lære meg Java. Jeg brukte derfor mye tid på å hente ut data fra mobiltelefonen. Dette førte til at det ble lagt mindre tid på kalmanfilteret. Feildisponering av tid er derfor årsaken til at kalmanfilteret ikke ble implementert ferdig. Dersom kalmanfilteret blir implementert ferdig forventes det måledata med mye høyre nøyaktighet. Det betyr man får brukbare måledata.

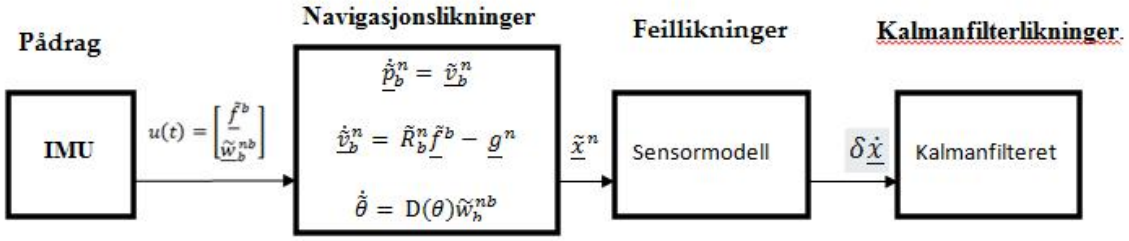
SensorDataRetriever applikasjonen, og alt arbeidet gjort i løpet av dette semesteret har kraftig økt min forståelse av utnyttelse av Android plattformen, og treghetsnavigering. Applikasjonen har lagt det fundamentale grunnlaget som trengs for å bygge den konkrete applikasjonen. Den er enkel å bruke, oversiktlig og ikke minst lagring av data i SD kortet gjør det enkelt å implementere dataene i Matlab. Noe som gjør det enkelt å analysere, og studere de rå dataene.

SensorDataRetriever applikasjonen for å logge data fra akselerometeret og gyroskopet, har vist å gi et godt resultat.Plottinger av rå data, beregnet- posisjon, hastighet og orienteringen i de forskjellige tilfellene (som ble presentert i kapitel 4) illustrerer nøyaktigheten på måledataene. Sensor dataene inneholder støy og bias feil.

5.2 Viderearbeid

Dataene fra *Magneticfield* sensoren blir ikke tatt i bruk på grunn av store unøyaktigheter. *SensorDataRetriever* applikasjonen logger dataene fra denne sensoren. Dersom sensorene i framtiden forbedres, anbefales det å inkludere disse dataene. Et mål for dette er å få bedre verdier fra sensoren slik at dataene kan brukes til å beskrive retningen mobiltelefonen beveger seg mot.

Et kalmanfilter bør implementeres.Hvis filteret fungerer optimalt vil dette kunne brukes for å estimere feil i sensorene. Da kan en komplett analyse av sensordataene bli utført. Strukturen på TNS med et kalmanfilter vises i figuren under.



Figur 5.1: Utvidet TNS blokkskjema for videre arbeid

Når en komplett analyse av brukbarheten av sensorene har blitt utført, og sensorene har de nøyaktighetene vi forventer kan man utvikle en applikasjon i Java som for eksempel beregner volumet av 3-dimensjonale objekter. Dette kan ha stor nytteverdi hos de fleste fraktselskapene.

Referanser

[1] Beskrivelse av komponentene i *IDE*

http://en.wikipedia.org/wiki/Integrated_development_environment (26.06.2011)

[2] Beskrivelse av komponentene i *Android SDK*

http://developer.android.com/sdk/adding-components.html (26.06.2011)

[3] Beskrivelse av *Android hardware*

http://developer.android.com/reference/android/hardware/Sensor.html (26.06.2011)

[4] Oddvar Hallingstad – *Matematiske mo*dellering av dynamiske systemer, (17.04.08)

[5] Anthony Lawrence – *Modern Inertial Technology:*
Navigation, Guidance, and Control (1992)

[6] Jørn Grahd- Masteroppgave: *Gyrokompass-nøyaktighet med MEMS-gyro (*06.06.2011)

[7] Beskrivelse av akselerometer

http://en.wikipedia.org/wiki/Accelerometer (26.06.2011)

[8] Beskrivelse av kompass sensoren (Magnetic Field Sensor) basert på MEMS teknologi

http://www.fisio.buap.mx/online/DrManjarrezE/Sensors1.pdf (26.06.2011)

[9] Beskrivelse av *DeadReckoning*

http://en.wikipedia.org/wiki/Dead_reckoning (26.06.2011)

[10] Oddvar Hallingstad – *Matematiske mo*dellering av dynamiske systemer
Kursnotat: *Eksempler på TNS-modeller*(03.03.2004)

[11] Oddvar Hallingstad – *Matematiske mo*dellering av dynamiske systemer
Kursnotat: *Standardmodeller og Kalmanfilterlikninger* (24.08.2005)

[12] Beskrivelse av struktur av en basis Android applikasjon (*Application Fundamentals*)

http://developer.android.com/guide/topics/fundamentals.html (26.06.2011)

Vedlegg

A Programmeringskode:

A.1 Java kode

A.1.1 SensorDataRetriever

```
package com.test.SensorDataRetriever;

import java.io.IOException;

// Implementering av androidkompnenter som ligger allerede i eclipse
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

// Entry classe
public class SensorDataRetriever extends Activity implements SensorEventListener,
OnClickListener{

 private TextViewlableAccelerometer, lableAccelerometerData;
 private TextViewlableGyroscope, lableGyroscopeData;
 private TextViewlableMagneticField, lableMagneticFieldData;
 private Button btnToggleLogging;
 private SensorManagermSensorManager;
 private Sensor mAccelerometer, mGyroscope, mMagneticField;
 private booleanhasAccelerometer, hasGyroscope, hasMagneticField;
 private long lastUpdateAccelerometer, lastUpdateGyroscope, lastUpdateMagneticField = -1;
private DataWriterlogWriter;

/** Called når aktiviteten kjøres første gang (onCreat). */
@Override
        public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main); // skjermen som hvises på telefonen
initUI();
registerListners();
   }
```

```
	private void initUI()
	{
		lableAccelerometerData = (TextView)findViewById(R.id.lableAccelerometerData);
		lableAccelerometer = (TextView)findViewById(R.id.lableAccelerometer);
		lableGyroscope = (TextView)findViewById(R.id.lableGyroscope);
		lableGyroscopeData = (TextView)findViewById(R.id.lableGyroscopeData);
		lableMagneticField = (TextView)findViewById(R.id.lableMagneticField);
		lableMagneticFieldData = (TextView)findViewById(R.id.lableMagneticFieldData);
		btnToggleLogging = (Button)findViewById(R.id.btnToggleLogging);
		btnToggleLogging.setOnClickListener(this);
	}

private void registerListners()
	{
	if(mSensorManager == null) //SensorManagerenhentesfraContext.getSystemService
	mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);

	// Prøver å sette opp en lytter 'listener' til akselerometeren
	if(mAccelerometer == null) //bruker SensorManageren til å hente 'default' akselerometer
	mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

	if(mAccelerometer != null) //Registererer en lytter på denne klassen med høy oppdatering frekvens
	hasAccelerometer = mSensorManager.registerListener(this, mAccelerometer,     SensorManager.SENSOR_DELAY_GAME);

	if(!hasAccelerometer) //hvis ikke den har verdi,så settes akselerometeren til 'Not Available'
	lableAccelerometer.setText("Accelerometer (Not Available)");

// Prøver å sette opp en lytter 'listener' til Gyroskopet
	if(mGyroscope == null) //bruker SensorManageren til å hente 'default' gyroskop
	mGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

	if(mGyroscope != null) //Registererer en lytter på denne klassen med høy oppdatering frekvens
	hasGyroscope = mSensorManager.registerListener(this, mGyroscope, SensorManager.SENSOR_DELAY_GAME);

	if(!hasGyroscope) //hvis ikke den har verdi,så settes gyroskopet til 'Not Available'
	lableGyroscope.setText("Gyroscope (Not Available)");

// Sette opp en lytter 'listener' til M.F.
	if(mMagneticField == null)  //bruker SensorManageren til å hente 'default' magnetisk felt
	mMagneticField = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

	if(mMagneticField != null) //Registererer en lytter på denne klassen med høy oppdatering frekvens
	hasMagneticField = mSensorManager.registerListener(this, mMagneticField, SensorManager.SENSOR_DELAY_GAME);

	if(!hasMagneticField) //hvis ikke den har verdi,så settes MagnetiskFeltet til 'Not Available'
	lableMagneticField.setText("Magnetic Field (Not Available)");

	}
```

```
@Override
	// SensorEventListener (1: onAccuracyChanged)
public void onAccuracyChanged(Sensor sensor, int accuracy) {
	if(hasAccelerometer&& sensor == mAccelerometer) //Akselerometeret finnes og har endret nøyaktighet
	lableAccelerometer.setText(String.format("Accelerometer ("+Helpers.AccuracyToString(accuracy)+")"));
	//Se 'Helpers filet'

	if(hasGyroscope&& sensor == mGyroscope) //Gyroskopet finnes og har endret nøyaktighet
	lableGyroscope.setText(String.format("Gyroscope ("+Helpers.AccuracyToString(accuracy)+")"));
	//Se 'Helpers filet'

	if(hasMagneticField&& sensor == mMagneticField) //Magnetiskfeltet finnes og har endret nøyaktighet
	lableMagneticField.setText(String.format("Magnetic Field ("+Helpers.AccuracyToString(accuracy)+")"));
	//Se 'Helpers filet'
	}

@Override
	//SensorEventListener (2: onSensorChanged)
public void onSensorChanged(SensorEvent event) {

if(hasAccelerometer&&event.sensor == mAccelerometer) //Akselerometeret finnes og endret verdier
		{
		logEvent(event);
		updateSensorData(lableAccelerometerData, lastUpdateAccelerometer, event.values);
		}

	if(hasGyroscope&&event.sensor == mGyroscope) // Gyroskopet finnes og har endret verdier
		{
		logEvent(event);
		updateSensorData(lableGyroscopeData, lastUpdateGyroscope, event.values);
		}

if(hasMagneticField&&event.sensor == mMagneticField) //Magnetiskfiltet finnes og endret verdier
		{
		logEvent(event);
		updateSensorData(lableMagneticFieldData, lastUpdateMagneticField, event.values)
		}

	}

@Override

protectedvoidonPause()
   {
	super.onPause(); // ikke log data hvis Appen er lukket (for å spare strøm og sånn ting)
	mSensorManager.unregisterListener(this);

	// Hvis vi driver å logge data til fil stoppe det
	stopLoggingData();
   }

@Override
protected void onResume()
	{
super.onResume();
registerListners();
   }
```

```
private void updateSensorData(TextViewtv, long lastUpdate, float[] values)
	{
		long currentTime = System.currentTimeMillis();

		if(lastUpdate == -1 || (currentTime - lastUpdate) > 100)
		{
			lastUpdate = currentTime;

			float x = values[SensorManager.DATA_X];
			float y = values[SensorManager.DATA_Y];
			float z = values[SensorManager.DATA_Z];

			tv.setText(String.format("x:%+2.5f, y:%2.5f, z:%2.5f", x, y, z));

		}
	}

@Override
	// Knappen i skjermen
public void onClick(View v) {

	if(logWriter != null)
	{
	stopLoggingData(); // hvis LogWriter har en verdi og knappen blir trukket stop logging av data
	}
	else
	{
	startLoggingData(); // hvis LogWriter har ikke en verdi og knappen blir trukket start logging av data
	}

}

private void startLoggingData() {

	if(logWriter != null)
	return;

	try { // hva skal filet hete: for eksempel log_20110603_051132
	logWriter = new DataWriter("Log_"+Helpers.TimeNow(Helpers.FILE_DATE_FORMAT_NOW),
"SensorDataRetriver");

	} catch (IOException e) {
		// Auto-generert catch blokk
		e.printStackTrace();
	}

		btnToggleLogging.setText(R.string.StopLogging);

	}

private void stopLoggingData() {

	if(logWriter == null)
		return;

	String fileName = logWriter.FileName;

	logWriter.Dispose();
	logWriter = null;
```

```
Toast toast = Toast.makeText(getApplicationContext(), "Log saved to "+fileName, Toast.LENGTH_SHORT);
                        toast.show();

            btnToggleLogging.setText(R.string.StartLogging);


            }

            private voidlogEvent(SensorEventevent)
            {
                        if(logWriter != null)
                                    logWriter.Write(event);
            }




}
```

A.1.2 DataWriter

```
package com.test.SensorDataRetriever;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.os.Environment;


public class DataWriter {

            private PrintWriter writer;
            private long timeStamp;
            private SensorDataaccelData, gyroData, magData;
            private booleanisDisposed = false;
            public final String FileName;

            public DataWriter(String fileName, String dirName) throws IOException
            {

            this.FileName = fileName;

            File dir;

            if(dirName.isEmpty())
            {
            // Hvis directory navnet er ikke spesifisert bruker vi rooten til SD kortet
```

```
        dir = Environment.getExternalStorageDirectory();
        }
        else
        {
        // .../ExternalStorage/dirName
        dir = new File(Environment.getExternalStorageDirectory().getAbsolutePath() +'/'+ dirName);

        // Hvisdirectroyeksistererikkeallerede, Create directory
        if(!dir.exists())
        dir.mkdir();
        }
        File file = new File(dir, fileName);
        file.createNewFile();

        writer = new PrintWriter(file);
        accelData = new SensorData();
        gyroData = new SensorData();
        magData = new SensorData();

        }

        public void Write(SensorEvent event)
        {
        Write(event.sensor.getType(), event.timestamp,
                event.values[SensorManager.DATA_X],
                event.values[SensorManager.DATA_Y],
                event.values[SensorManager.DATA_Z]);
        }
        public void Write(intSensorType, long timeStamp, float x, float y, float z)
        {
                switch(SensorType)
                {
                case Sensor.TYPE_ACCELEROMETER:
                accelData.SetValues(x, y, z);
                        break;
                case Sensor.TYPE_GYROSCOPE:
                gyroData.SetValues(x, y, z);
                        break;
                case Sensor.TYPE_MAGNETIC_FIELD:
                magData.SetValues(x, y, z);
                        break;
                default:
                        return; // Ignorer alt annet
                }

                this.timeStamp = timeStamp;
                writeDataToFile();
        }

private voidwriteDataToFile()
        {
                if(isDisposed) // Ikke skriv noe hvis writer er lukket.
                return;

                if(accelData.HasValues() &&gyroData.HasValues() &&magData.HasValues())
                {
                char seperator = ';';

                writer.print(this.timeStamp);
```

```
                writer.print(seperator);
                printData(accelData, seperator);
                writer.print(seperator);
                printData(gyroData, seperator);
                writer.print(seperator);
                printData(magData, seperator);
                writer.println();
                }

        }

        private voidprintData(SensorData data, charseperator)
        {
                // Printer <data.x><seperator><data.y><seperator><data.z>
                writer.print(data.x);
                writer.print(seperator);
                writer.print(data.y);
                writer.print(seperator);
                writer.print(data.z);
        }


        public void Dispose()
        {
                isDisposed = true;
                writer.close();
        }


        private class SensorData
        {
                public java.lang.Float x, y, z = null;

                public SensorData()
                {

                }
                public void SetValues(float x, float y, float z)
                {
                        this.x = x;
                        this.y = y;
                        this.z = z;
                }

                public booleanHasValues()
                {
                        // Hvis ingen av feltene er "null"
                        return x != null && y != null && z != null;
                }
        }
}
```

A.1.3Helpers

```
package com.test.SensorDataRetriever;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import android.hardware.SensorManager;

public class Helpers {
	public static final String FILE_DATE_FORMAT_NOW = "yyyyMMdd_HHmmss";

	public static String AccuracyToString(int accuracy)
	{
		String str;

		switch(accuracy)
		{
		case SensorManager.SENSOR_STATUS_UNRELIABLE:
			str = "Unreliable";
			break;
		case SensorManager.SENSOR_STATUS_ACCURACY_LOW:
			str = "Low";
			break;
		case SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
			str = "Medium";
			break;
		case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
			str = "High";
			break;
		default:
			str = "Unknown";
		}

		return str;
	}

	public static String TimeNow(String format)
	{
		DateFormatdateFormat = new SimpleDateFormat(format);
		return dateFormat.format(new Date());
	}

}
```

A.2 Matlab–kode

A.2.1 IV_TNS

```
%% IV_TNS

clc, clear all, close all;

%a=importdata('loggedataAG.txt',' ');
a=importdata('logg25.mat');
%a=importdata('bev0grad.txt');
[N,Ns]=size(a);

t = 0;
p_b_n =     zeros(3,N);            % Start posisjon
v_b_n =     zeros(3,N);            % Start Hastighet
th =         zeros(3,N);            % Start Orientering
x=           [p_b_n; v_b_n; th];  %Start x
u =          zeros(6,N);

t0=a(1,1);
ta=(a(:,1)-a(1,1))*(10^-9);
t=ta;
xa=a(:,2);
ya=a(:,3);
za=a(:,4);
f__b = [a(:,2)'; a(:,3)'; a(:,4)']; % Spesfikk kraft

xg=a(:,5);
yg=a(:,6);
zg=a(:,7);
w_b_nb = [a(:,5)'; a(:,6)'; a(:,7)']; %Vinkelhastighet


%% Heun's metode

for k = 1:N-1

    h(k) = t(k+1)-t(k);
u(:,k) = [f__b(:,k);w_b_nb(:,k)];
u(:,k+1) = [f__b(:,k+1);w_b_nb(:,k+1)];
    xm(:,k+1) = x(:,k) + h(k)*test(x(:,k),u(:,k),0);
x(:,k+1) = x(:,k)+(h(k)/2)*((test(x(:,k),u(:,k),0))+
      (test(xm(:,k+1),u(:,k+1),0)));

end
```

A.2.2funksjon

```
function f = test(x,u,t)
%IV_INS;

%t = 0:0.2:50;
p_b_n =      [x(1);x(2);x(3)];     %Posisjonen
v_b_n =      [x(4);x(5);x(6)];     %Hastigheten
th =         [x(7);x(8);x(9)];     %Vinkel
f__b =        [u(1);u(2);u(3)];     %Spesifikk kraft
w_b_nb =      [u(4);u(5);u(6)];     %Vinkelhastighet
g =          9.81;               % Gravity
g_n =        [0;0;g];

R_1=            [1 0 0;
                    0 cos(x(7)) -sin(x(7));
                    0 sin(x(7)) cos(x(7))];

    R_2=            [cos(x(8)) 0 sin(x(8));
                    0 1 0;
                    -sin(x(8)) 0 cos(x(8))];

    R_3=            [cos(x(9)) -sin(x(9)) 0;
                    sin(x(9)) cos(x(9)) 0;
0 0 1];


   R_b_n = R_3*R_2*R_1;

D = [1 (sin(x(7))*tan(x(8))) (cos(x(7))*tan(x(8)));
            0 cos(x(7)) -sin(x(7));
            0 (sin(x(7))/cos(x(8))) (cos(x(7))/cos(x(8)))];


f_1 = v_b_n(:);
f_2 = (R_b_n*f__b(:))-g_n;
f_3 = D*(R_b_n*w_b_nb(:));

f = [f_1;f_2;f_3];
```

A.2.3 Kalmanfilteret

```
%% Kalmanfilteret
clc, clear all, close all;

%a=importdata('loggedataAG.txt',' ');
a=importdata('logg25.mat');
[N,Ns]=size(a);

% Tid med 0 som startpunkt
Time=(a(:,1)-a(1,1))*(10^-9);
Ts=[Time(2:end)]-Time(1:(end-1));

Acc_body=a(:,2:4)';
Gyr_body=a(:,5:7)';
clear a;

% init x:
xe=zeros(15,N);
xp=zeros(15,N);
R=0.0;    % Målestøy
Q=0.0;     % Prosesstøy
pe=eye(15);%intit kovar
H=[zeros(15,3) eye(15,3) zeros(15,3) zeros(15,3) zeros(15,3)];
for i = 2:N

% Henter inn diskretiserte system matriser
    [Phi,Ga] = sysmatriser(xe(:,i),Acc_body(:,i),Ts(i-1));

    xp(:,i+1)=Phi*xe(:,i);
    pp=Phi*pe*Phi' + Ga*Q*Ga';

    z(1:15,i)=0;% Her må du hetne målinger!!!
    K=pp*H'*(H*pp*H'+ R)^(-1);
    xe(:,i+1)=xp(:,i+1)+ K*(z(:,i)-H*(xp(:,i)));
pe=(eye(size(K*H))-K*H)*pp;
end
```